Evolutionary Algorithms

with Mixed Strategy

Liang Shen

Supervisors: Dr. Jun He
Prof. Qiang Shen

Ph.D. Thesis

Department of Computer Science
Institute of Mathematics, Physics and Computer Science
Aberystwyth University

April 16, 2016

Declaration and Statement

DECLARATION

This work has not previously been accepted in substance for any degree and is not being concurrently submitted in candidature for any degree.

Signed ..... (candidate)

Date .....

STATEMENT 1

This thesis is the result of my own investigations, except where otherwise stated. Where **correction services**[1] have been used, the extent and nature of the correction is clearly marked in a footnote(s).

Other sources are acknowledged by footnotes giving explicit references. A bibliography is appended.

Signed ..... (candidate)

Date .....

STATEMENT 2

I hereby give consent for my thesis, if accepted, to be available for photocopying and for inter-library loan, and for the title and summary to be made available to outside organisations.

Signed ..... (candidate)

Date .....

[1] This refers to the extent to which the text has been corrected by others.

Abstract

During the last several decades, many kinds of population based Evolutionary Algorithms have been developed and considerable work has been devoted to computational methods which are inspired by biological evolution and natural selection, such as Evolutionary Programming and Clonal Selection Algorithm.

The objective of these algorithms is not only to find suitable adjustments to the current population and hence the solution, but also to perform the process efficiently. However, a parameter setting that was optimal at the beginning of the algorithm may become unsuitable during the evolutionary process. Thus, it is preferable to automatically modify the control parameters during the runtime process. The approach required could have a bias on the distribution towards appropriate directions of the search space, thereby maintaining sufficient diversity among individuals in order to enable further ability of evolution.

This thesis has offered an initial approach to developing this idea. The work starts from a clear understanding of the literature that is of direct relevance to the aforementioned motivations. The development of this approach has been built upon the basis of the fundamental and generic concepts of evolutionary algorithms.

The work has exploited and benefited from a range of representative evolutionary computational mechanisms. In particular, essential issues in evolutionary algorithms such as parameter control, including the general aspects of parameter tuning and typical means for implementing parameter control have been investigated. Both the hyper-heuristic algorithm and the memetic algorithm have set up a comparative work for the present development. This work has developed several novel techniques that contribute towards the advancement of evolutionary computation and optimization.

One such novel approach is to construct a mixed strategy based on the concept of local fitness landscape. It exploits the concepts of fitness landscape and local fitness landscape. Both theoretical description and experimental investigation of this local fitness landscape âĂŞbased mixed strategy have been provided, and systematic comparisons with alternative approaches carried out. Another contribution of this thesis is the innovative application of mixed strategy. This is facilitated by encompassing two

mutation operators into the mixed strategy, which are borrowed from classical differential evolution techniques. Such an improved method has been shown to be simple and easy for implementation.

The work has been utilised to deal with the problem of protein folding in bioinformatics. It is demonstrated that the proposed algorithm possesses an appropriate balance between exploration and exploitation. The use of this improved algorithm is less likely to fall into local optimal, entailing a faster and better convergence in resolving challenging realistic application problems.

Acknowledgements

Foremost, I would like to express my deep and sincere gratitude to my supervisor Dr Jun He for his inspiration, patience and encouragement. He introduced me to the field of Evolutionary Algorithm and provided lots of good ideas and advice. I would have never completed this thesis without his help. He educated me generously in how to be a good researcher with his immense knowledge. The joy and enthusiasm he has for his research were contagious and motivational for me to hurdle all the obstacles.

Besides Jun, I am deeply grateful to my second supervisor, Prof Qiang Shen, for his insightful comments and hard questions throughout my PhD study. His great ideas and unfailing support guided me to go further in this field. His wide knowledge and logical way of thinking have been of great value for me.

Luckily for me, I have benefited from numerous research fellows who turned into even better friends in the Advanced Reasoning Group, especially to Xin Fu, Longzhi Yang, Ren Diao, Chengyuan Chen, Pan Su, Tianhua Chen. They gave me wise advice and helped me along the way. Also, I am indebted to all the colleagues and the staff in the Department of Computer Science at Aberystwyth University for providing a stimulating and fun environment in which to learn and grow. I am especially grateful to Dr Neil S. Mac Parthaláin, Peter Scully, Jinping Song, Ling Zheng, Zhengpeng Li and Nitin Kumar Naik for their support, company and laughs.

I would like to thank my best friend Xiao Liu, for helping me get through the most difficult times, and for all the emotional support, entertainment, and caring they provided.

Last but not the least, I would like to thank my family: my parents Xianwen Zeng, Ping Shen and Wei Zeng, for all their love and encouragement. They supported me in all my pursuits and put me where I am today. To them I dedicate this thesis.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

There are a large number of problems existing in our real life that behave as obstacles, which people need to solve them to get to the next stage. People are always willing to solve them in the easiest way. However, in most of time, problems are quite different, and even the same problem may demand different solutions when we encounter it in variable situations. It may require a great deal of effort by only considering one approach or **strategy**, which might be most conventional one. In light of this, people try to consider the nature of the problem itself, carefully setting the target, the **Objective**, and looking for the most suitable way to tackle it.

With an investigation of the problem's nature characteristic, people try to find the exact strategy which exactly fits to the problem. But this is only the most desired situation, which may require people devoting unexpected huge time when facing different problems. Therefore, it would be more efficient to categorise problems into a selection of different types, and design the most suitable strategy for certain category. By allocating different strategy to each categories, a reserved pool of strategies are created.

1.1 Evolutionary Algorithms

Evolutionary algorithms are a group of bio-inspired algorithms often employed as optimisation techniques that attempt to search for the best solution it could find, if not the optimal one, to a difficult problems which are conventionally time-consuming problems if using a greedy search. It is the process of repeatedly generating a population

of solutions to the target problems, of which unfavourable solutions with worse result or unstable performance are eliminated. By iterating the process including generation of potential solutions, such as mutation and crossover, selection of solutions by removing unhelpful ones, and adjustment to automatically adapt to the current situation, significant computing time can be saved and models can also be built to generate better solutions.

1.2 Mixed Strategy

The aim of this project is to develop a self-adaptive method that draws from game theory to make evolutionary algorithms get better performance to varied problems. There have been many approaches of adjusting evolutionary algorithms, but most of these have been focused on specific applications and thus tend to address one single pure task. It is interesting to explore a mixed strategy that can support not only continuous domain, primarily on numeric optimization, but also discrete problems, while these issues are usually conducted in rather different areas. It is important that this is to be examined logically to what extent the mixed strategy can adapt for the algorithms that could support learning.

This project is set to investigate and understand the main design issues that may be involved in integrating significant different algorithms, including different types of evolutionary algorithms. This will help make clear any underlying assumptions and exposing any essential conditions upon which to successfully develop a robust and efficient mechanism, the mixed strategy, whereby different problems could be solved follow this idea.

The performance of evolutionary algorithms is affected by many factors (e.g. mutation operators and selection strategies). Take Evolutionary Programming (EP) as an example, although the conventional approach with Gaussian mutation operator may be efficient, the initial scale of the whole population can be very large. This may lead to the conventional EP taking too long to reach convergence. To combat this problem, EP has been modified in various ways. In particular, modifications of the mutation operator may significantly improve the performance of EP.

Several mutation operators, Gaussian, Cauchy and Lévy mutations [1][2][3] have been developed in evolutionary programming (EP in short). However, according to the

no free lunch theorem [4], none of them is efficient in solving all optimization problems. In other words, each mutation is efficient only for some specific fitness landscapes. Experiments have also confirmed this point. For example, Gaussian mutation has a good performance for some uni-modal functions and multi-modal functions with only a few local optimal points; Cauchy mutation works well on multi-modal functions with many local optimal points [1].

An approach to improve conventional EP using single a mutation operator is to apply different mutation operators simultaneously and integrate their advantages together. Such a strategy can be called a mixed mutation strategy in terms of game theory. There are different ways to design a mixed strategy. For example, an early implementation is a linear combination of Gaussian and Cauchy distributions [5]. Improved fast EP (IFEP) [1, 3] takes on another technique: Each individual implements Cauchy and Gaussian mutations simultaneously and generates two individuals; the better one will be chosen in the next generation. The advantage of these two mixed mutation strategies is their simplicity in implementation. A mixed strategy is inspired from game theory [6],[7], which chooses a mutation operator according to a probability distribution. Reinforcement learning theory is also used to learn individual mutation operators [8].

In previous studies [7][9], the design of a mixed strategy mainly utilizes the reward of each operator (i.e. an operator which produces a higher fitness will receive a better reward), but little is relevant to the fitness landscape. However, the performance of each mutation operator is strongly linked to the the fitness landscape, so it is important to deal with the local fitness landscape where an population is located.

In this project the mixed strategy is proposed to adapt to the local fitness landscape. Firstly a measure about the local fitness landscape on the multi-modality is introduced; and then the new mixed strategy is adjusted with respect to the above measure value.

1.3 Thesis Structure

This section outlines the structure of the remainder of the thesis.

Chapter 2: This chapter provides a background overview of the literature directly relevant to the work carried out in the subsequent chapters. First, it briefly introduces

the fundamental and generic concepts of evolutionary algorithms, including the basic representation of population and individuals and the commonly used key evolutionary computational operations such as mutation, crossover and selection. This is followed by a description of representative evolutionary computational mechanisms, including evolutionary programming, ant colony optimization, and clonal immune algorithm from artificial immune systems. The chapter then addresses the essential issue of parameter control in evolutionary algorithms, examining the general aspects of parameter tuning and typical means for implementing parameter control. It then moves on to the technical aspects of self-adaptive parameter control where the idea of combining different algorithms together is shown. This hybrid algorithm is conceived to be self-guided that is able to choose the right method in a given situation. Two established algorithms on this idea, hyper-heuristic algorithm and memetic algorithm are then introduced later in the chapter. Finally, the review focuses on the motivation and the use of mixed strategies in evolutionary algorithms, setting the foundation for the following developments.

Chapter 3: This chapter presents a novel approach to constructing a mixed strategy based on the concept of local fitness landscape. It first introduces the underlying concept of fitness landscape, and its local version. Then, the chapter develops a novel mixed strategy that strengthens conventional evolutionary programming with two important improvements: a) applying local fitness landscapes to aid in the determination of the behaviour of mutations in evolutionary programming; and b) proposing a training procedure that makes use of typical learning functions to determine the preferable probability distribution of mixed mutation operators, in response to various types of local fitness landscape. Both theoretical description and experimental investigation of these are given. Systematic comparisons with alternative approaches are carried out, supported with an analysis of the experimental results. The results demonstrate that the proposed approach successfully addresses and therefore, avoids a number of major drawbacks of using conventional evolutionary programming methods that employ a single mutation operator.

Chapter 4: This chapter presents a different approach to the development of a mixed strategy, by exploiting game theory with the use of incomplete information. The work results in a modified mixed strategy which combines different mutation operators. This new approach is compared to the strategy shown in the preceding

chapter through systematic experimental evaluation, using the same test functions previously adopted. The results once again demonstrate that this newly introduced algorithm can successfully combat the shortcomings of conventional evolutionary programming methods that employ a single mutation operator. The new approach has proven to perform at least as well as the best of different conventional strategies with single mutations. Furthermore, the test results also illustrate that the approach enables a more stable performance while in use.

Chapter 5: This chapter presents an innovative application of mixed strategy by extending the domain of usage of mixed strategy to discrete problems. Given similar features to those associated with numerical function optimisation that is based on the different types of local fitness landscape, this work applies the mixed strategy to immune algorithms, by encompassing two mutation operators borrowed from classical differential evolution techniques. This leads to a potentially powerful optimisation algorithm with simple and easy implementation. The work is applied to addressing the problem of protein folding in bioinformatics. Experimental results demonstrate that the proposed algorithm possesses an appropriate balance between exploration and exploitation, such that it is less likely to fall into local optimal and has a faster and better convergence, in resolving challenging realistic application problems.

Chapter 6: This chapter summarises the achievements of the work carried out in the preceding chapters and points out lessons learned so far in developing evolutionary algorithms with mixed strategy and their application. It also discusses possible improvements over the present research, including ideas for both long-term and short-term developments.

Chapter 2

Background and Literature Review

Since the advent of the exploitation of bio-inspired computation and implications, it is seen an significantly increasing interest in research on the utilization of those novel bio-inspired technologies in the context of designing effective optimisation procedures for the important components of those more complex recognition problems. A particularly successful domain in recently rising research interests, as previously stated within the categorizing framework of meta-heuristic algorithms, is the application of evolutionary computation in optimisation, both in continuous problems and discrete ones. Evolutionary algorithms are usually reported to deliver good results, but exceptions have been reported where simpler (and faster) algorithms result in higher accuracy on particular data sets.

In particular, the exploitation of bio-inspired computation has given rise to the probability of advancing optimisation techniques. It is owing to this observation: evolutionary algorithm methods are considered herein to serve the foundation upon which to develop hybrid algorithms for optimisation domain. A brief overview of the basic evolutionary computation mechanisms is provided below.

2.1 Evolutionary Algorithms

Upon the their first introduction, evolutionary algorithm techniques have increasingly grown as a problem solving mechanism based on the principle of evolution. Systems

built using such techniques typically maintain a population of potential solutions. They all employ a certain selection process based on the fitness of the individual solutions, and certain "genetic" operators [10]. There are several different types of such systems. The three most popular are genetic algorithms (GAs) [11], evolutionary programming (EP) [12] and evolution strategies (ESs) [13] [14]. Genetic algorithms are usually reported to deliver good results, but exceptions have been reported where simpler (and faster) algorithms result in higher accuracy on particular data sets. Evolutionary computation differs from traditional optimization techniques in that it involves a parallel search through the population of solutions.

2.1.1 Mechanism of Evolutionary Algorithms

An evolutionary algorithm is a stochastic procedure which maintains a population of individuals for a potential iteration t, $P(t) = \{x_1^t, \ldots, x_n^t\}$. Each individual $x_i (i = 1, 2, \ldots, n)$ represents a potential solution to the given problem, and, the individual is implemented using a certain data structure S which could possibly be rather complex. Each solution x_i^t is evaluated to measure its quality, namely the "fitness". Then, the system generates the population (iteration $t+1$) anew with the aid of selecting the more fit individuals subject to an evaluation function (select step).

Some randomly chosen members of the new population undergo transformations (alter step) by means of genetic operators to form new solutions, the offspring. In implementing the alter step, there are unitary transformations m_i (mutation type), which create new individuals by a small change, a flip between number 1 and 0 in conventional evolutionary, in a single individual, $(m_i : S \to S)$ and higher order transformations c_j (crossover type), which create new individuals by combining parts from several (two or more) individuals $(c_j : S \times \ldots \times S \to S)$. After a certain number of generations, the program would come to a convergence. The best individual then is supposed to represent an acceptable near-optimum solution.

2.1.2 Representation of individuals

Genetic algorithms started and still mainly operate with binary strings for representing individuals, that is, their genotype. If the evaluation function is not pseudo-Boolean,

each string has to be decoded into a set of appropriate decision variables, namely the phenotype, before the fitness of the individuals can be evaluated.

Evolution strategies started with integer variables as an experimental optimum-seeking method, but turned to real variables when used in practical problem-solving systems. The individuals are not only represented by the set of decision or object variables, but also by a set of strategic parameters controlling the variation process, i.e. variances and covariances. This latter parameter set is learned on-line during the search for optima.

EP in its current form relies upon real variables, both for the object variables and the strategy parameters, which are adapted according to exogenous rules.

2.1.3 Genetic Operators

Following the construction of the problem domain into a string of variables, the initialisation step then generates a group of these strings (individuals), typically containing several hundreds or thousands of them. They collectively form the entire population of potential solutions. Then the generation of the next generation population of solutions is carried out on the next step, through a combination of genetic operators, most importantly, including mutation and crossover.

Mutation

In the bit-string world of genetic algorithms, mutations are purely random bit inversions, occurring with low frequencies generally. Evolution strategies and evolutionary programming both use Gaussian noise with zero mean to perturb all object variables. Evolution strategies additionally assumes logarithmic normal distribution for the standard deviations of the mutation step sizes and normal distributions for changing the covariances which may lead to correlated mutations.

Crossover

It is very interesting to observe that genetic algorithms emphasize the role of recombination, e.g. in the form of two-point or multi-point crossover, whereas EP rejects this form of variation as useless or sometimes even harmful. An explanation can be found if probability distributions are examined for changes of the object variables at the level

of phenotypes, i.e., after decoding the bit strings. Indeed, crossover in GAs may lead to recombinant which lie outside of the hypercube spanned by their parental positions.

Evolution strategies rely on both mutation and recombination. In particular, discrete recombination is similar to uniform crossover if the crossover points lie on the boundaries of the partial bit strings, which encode the different phenotype object variables. Intermediate recombination, recommended for strategy parameters, helps to avoid overadaptation, but may lead to a loss of diversity of internal models of the individuals and must be counterbalanced by mutation.

2.1.4 Selection

The most striking differences exist between genetic algorithms and evolutionary programming on the one hand and evolution strategies on the other hand with respect to the selection procedures. However, it is not merely the scheme of assessing the individuals for their fitness that plays a role here. Two other processes are intermingled the generation transition and the mating behaviour.

If elitist variants are excluded, namely good parental positions cannot get lost, which is good for proving global convergence, all three classes of canonical evolutionary computation methods give their individuals a life span of one generation. In general, Genetic algorithms and evolutionary programming produce just one descendant on average per generation. This is true for genetic algorithms with crossover as well, since only one of the two recombinant is used later on, generally at random, i.e. without comparing fitness. Only evolution strategies operate with a surplus of descendants, with the $(\mu; \lambda)$ version where $\mu(> \lambda)$ children are reproduced from λ parents. This helps in handling inequality constraints, the violation of which leads to infeasible descendants.

With proportional selection as well as most other forms like (linear) ranking, all individuals produced during generation t within genetic algorithms and evolutionary programming have a chance to have offspring themselves in the next generation $t+1$. Evolution strategies, however, discard the $\lambda - \mu$ worst descendants. The remaining μ individuals become parents of the next generation and possess equal chances to mate and have children. Genetic algorithms and evolutionary programming allocate mating as well as reproduction probabilities to their individuals according to the relative fitness values or the relative position in the ranking process.

2.2 Examples of Evolutionary Algorithms

2.2.1 Evolutionary Programming

Evolutionary programming (EP) is a branch, alongside other notable research areas such as genetic algorithms and evolution strategy, of evolutionary computation that stems from natural biological evolution [12]. Evolutionary programming operates on the basis of populations. The objective is not only to find suitable adjustments to the current population and hence the solution, but also to perform the process efficiently.

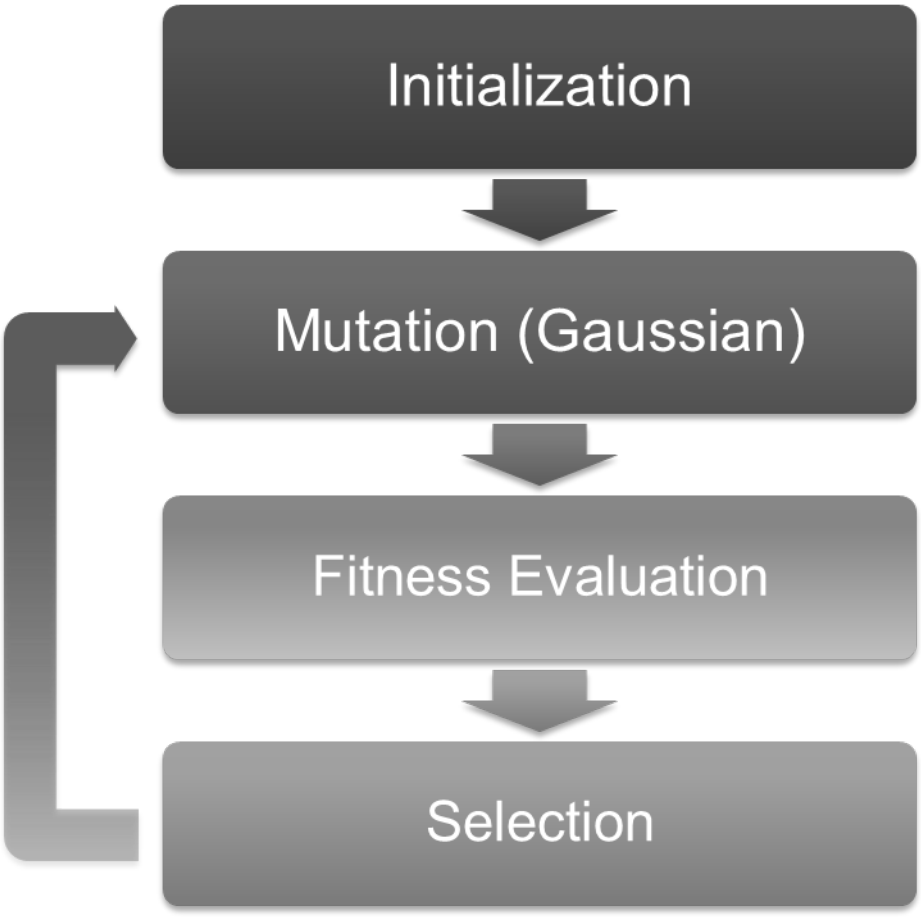

Figure 2.1: General procedure of evolutionary programming

Basic Operations of Evolutionary Programming

Evolutionary programming is a powerful algorithm for numerical optimization [1], where it is used to find a minimum $\vec{x}_{\min}$ of a continuous function $f(\vec{x})$, that is,

$$f(\vec{x}_{\min}) \leq f(\vec{x}), \quad \vec{x} \in D, \tag{2.1}$$

where D is a hypercube in $\mathbb{R}^n$, n is the dimension.

The general procedure of conventional evolutionary programming uses a single mutation operator is shown on Fig. 2.1, which can be described as follows [1]:

1. **Initialization:** Generate an initial population consisting of μ individuals at random. Each individual is represented by a set of real vectors $(\vec{x}_i, \vec{\sigma}_i)$, for, $i = 1, \cdots, \mu$

$$\vec{x}_i = (x_i(1), x_i(2), \cdots, x_i(n)),$$

$$\vec{\sigma}_i = (\sigma_i(1), \sigma_i(2), \cdots, \sigma_i(n)).$$

2. **Mutation:** For each parent $(\vec{x}_i^{(t)}, \vec{\sigma}_i^{(t)})$ (where t represents generation), create an offspring $(\vec{x}_i', \vec{\sigma}_i')$ as follows: for $j = 1, \cdots, n$,

$$\sigma_i'(j) = \sigma_i^{(t)}(j) \exp\{\tau N(0,1) + \tau' N_j(0,1)\}, \tag{2.2}$$

$$x_i'(j) = x_i^{(t)}(j) + \sigma_i^{(t+1)}(j) X_j, \tag{2.3}$$

   where $N(0,1)$ stands for a Gaussian random variable generated for a given i, $N_j(0,1)$ is a Gaussian random variable generated for each j, and X_j is a random variable generated for each j. The parameter τ' controls the global search step size, whereas τ is the factor for individual search step size. They are chosen as the same as in [1]:

$$\tau = (\sqrt{2\sqrt{n}})^{-1},$$

$$\tau' = (\sqrt{2n})^{-1}.$$

3. **Fitness Evaluation:** For μ parents and their μ offspring, calculate their fitness values $f_1, f_2, \cdots, f_{2\mu}$.

4. **Selection:** Define and initialize a winning function for every individual in parent and offspring population as $w_i = 0$, $i = 1, 2, \cdots, 2\mu$. For each individual i, select one fitness function, say f_j and compare the two fitness functions. If f_i is less than f_j, then let $w_i = w_i + 1$. Perform this procedure q times for each individual.

   Select μ individuals that have the largest winning values to be the parents of the next generation.

5. Repeat steps 2-4, until the stopping criteria are satisfied. The stopping criteria defines the termination condition of the running of the process. It is set to a fixed number of generations reached in this algorithm. The number should be large enough such that the process is usually not able to produce significantly better results.

Thus, the general process of Evolutionary Programming includes four major steps: Initialization, Mutation, Fitness Evaluation and Selection, as shown in Fig. 2.1.

In addition, to avoid the step size σ falling too low, a lower bound $\sigma_{\min}$ should be put on σ [15]. So a revised scheme of updating σ is given by:

$$\sigma_i'(j) = (\sigma_{\min} + \sigma_i^{(t)}(j)) \exp\{\tau N(0,1) + \tau' X_j\}.$$

where $\sigma_{\min} > 0$ is the minimum value of step size σ.

2.2.2 Ant Colony Optimization

Ant Colony Optimization (ACO) was initially introduced in the early 1990's as a novel nature-inspired metaheuristic for the solution of hard combinatorial optimization (CO) problems [16, 17]. This algorithm is based on the behaviour of real ant colonies in which ants are capable of finding the shortest route between a food source and that, more significantly, adapting to changes in the environment.

Informally, the ACO algorithm can be summarized as follows: A group of ants randomly search the space surrounding their nest in order to explore a location rich in foods. The origin of the ability of ants to find out the shortest routes to any foods source lies in the deposits of the chemical pheromone. As soon as an ant finds a food source, it

evaluates the quantity and the quality of the food and carries some of it back to the nest. During the return trip, the ant deposits a chemical pheromone trail on the ground. The quantity of pheromone deposited (and also evaporated) over time, which may depend on the quantity and quality of the food as well as the number of ants, will guide other ants to the food source.

Computationally, an ant is a simple computational agent, which iteratively constructs a solution to the given problem. Partial problem solutions are seen as states. At the core of an ACO algorithm lies a loop, where at each iteration, each ant moves from a state ι to another Ψ, corresponding to a more complete partial solution. That is, at each step σ, each ant k computes a set $A_k^\sigma(\iota)$ of feasible expansions to its current state, and moves to one of these in probability. The probability distribution is specified as follows. For ant k, the probability $p_{\iota\Psi}^k$ of moving from state ι to state Ψ depends on the combination of two values [18]:

- The attractiveness η of the move, as computed by some heuristic indicating the desirability of that move *apriori*; and
- The trail τ of the move, indicating how proficient it has been in the past to make that particular move: it is therefore indicative of the desirability of that move a posteriori.

Trails are updated usually when all ants have completed their solution, increasing or decreasing the level of trails corresponding to moves that are part of "good" or "bad" solutions, respectively.

2.2.3 Immune Algorithms

Artificial immune system (AIS) has emerged as a biologically-inspired approach that imitated the human immune systems for solving various types of computational problems such as optimization, classification and a large variety of real-world applications [19, 20]. Over the last decade of the development, the research of AIS has been divided into three types of models: clonal selection, negative selection theories, and immune networks [21, 22].

2.2.3.1 The Biological Immune System

The immune system protects an animal from being attacked by foreign micro-organisms. When an antigen (virus, bacteria etc.) first exposed to the system, it can extract information from the antigen and use that information in future cases of re-infection by the same or similar antigens. From a computational point of view, this ability makes the immune system useful [19].

The immune system can be divided into two tiers of defence: the innate immune system and the adaptive immune system. In the innate immune system, granulocytes and macrophages play a mediate role. With these two cells the system can immediately fight antigens without requiring any previous exposure to them. Some cells in the innate immune system ingest and digest micro-organisms and antigenic particles and some mediate interactions between the antigen and other immune cells [19].

In the adaptive immune system, the mediate cells called lymphocytes. B cells and T cells are the two types of it. There is a principle called clonal selection principle or clonal expansion principle, which describes the response of the adaptive immune system to antigens. The theory of it is that only cells that can recognise the antigen are selected and clone. Both B-cells and T-cells undergo clonal expansion, but only B cells experience somatic mutation. It is because B cells is mutation of the gene region responsible for recognising antigens [19].

When a mammal is exposed to an antigen, the B cells produce antibodies. The B cell is stimulated by the antigen binding with its receptors and by signals from other immune cells. With this stimulation, the B cells proliferate (clone) and most of the new cells mature into non-dividing plasma cells. Some mature cells became B-memory cells that circulate through the blood and tissues. When it is exposed to the antigen again, plasma cells proliferate with high antigen affinity [19].

2.2.3.2 Clonal Selection Algorithm

A large proportion of studies in AIS have been focused on clonal selection algorithm which could be utilized as efficient algorithm for optimization problems [23]. Although initially thought by someone as genetic algorithm (GA) without crossover [24], clonal selection algorithm, with the features of affinity proportional reproduction and hypermutation, has been seen as a robust algorithm in the research field of Evolutionary

Algorithm (EA) alongside other approaches, such as genetic algorithm and swarm intelligence algorithm [25, 26].

The clonal selection theory [27] in an immune system describes the phenomenon that the immune system performs a natural response when binding an antigenic stimulus, where the B cells are able to recognize the antigens, and start to proliferate to provide solution to the antigens. Several types of algorithms such as CLONal selection ALGorithm (CLONALG) [26] and optimization Immune Algorithm (opt-IA) [28], have been proposed to tackle the optimization problems using the basic processes involved in clonal selection.

Basic Operations of Clonal Selection Algorithm

The basic idea of Clonal Selection Algorithm (CSA) involves two populations: a population of antigens, and a population of antibodies, where the antigens represent the problems to be solved, and the antibodies are the current candidate solutions

1. **Initialization**: A basic process of CSA starts with a randomly initialization of the population of individuals (M). The affinity (fitness function value) of all antibodies (individuals) in population M are determined with respect to the antigens (the given objective function).

2. **Cloning**: The cloning operator will then select n best individuals with highest affinity from population M and generate n copies of these individuals proportionally to their affinity with the antigen, forming the clone population P^{clo}. The higher the affinity, the higher the number of P^{clo}, and vice-versa.

3. **Mutation**: Then the hypermutation operator performs mutation to all these n individual in P^{clo} with a rate inversely proportional to their fitness values, generating the P^{hyp}. After computing the affinity of the antigen, CLONALG randomly creates new antibodies that replace the antibodies with lowest fitness in the current population.

4. Afterwards, the algorithm repeat these process until the stopping criteria terminates the iteration, which is typically a predefined number of generations is reached.

2.3 Parameter Control in Evolutionary Algorithms

In applying any heuristic search algorithm like evolutionary algorithm, there are basically two major steps under consideration, one being the choice of representation and the other being the fitness function. Based on the specification of representation and fitness function, one can go on to determine which component should be technically required or better fitted in for the chosen representation and fitness function. As for evolutionary algorithm, one would typically consider components such as mutation/recombination operators for its representation, selection strategy for selecting parents, survivors as well as initial population. Each component may have parameters involved, which would greatly influence the performance as to whether a more optimal solution will be achieved or a solution will be find within efficient time. However, finding such parameters have long been challenging yet promising task among evolutionary algorithms researchers and practitioners.

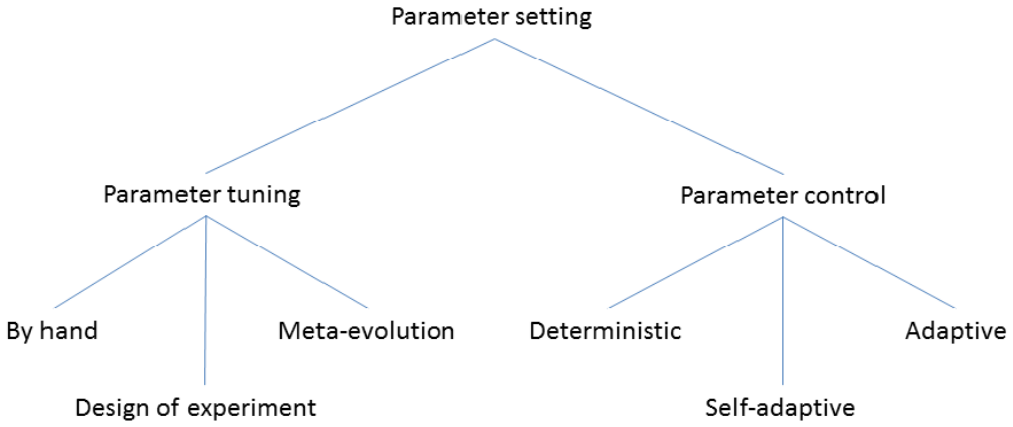

Figure 2.2: Taxonomy of different parameter setting methods

In [29], Eiben et al. successfully introduced a taxonomy for classification of different parameter setting methods, which we will adopt in this thesis as shown in figure 2.2. On the topmost level they firstly separate all kinds of parameter strategies into 2 branches, namely parameter tuning and parameter control, which is based on an early taxonomy of Angeline [30].

2.3.1 Parameter Tuning

By parameter tuning we mean that parameters involved in a particular evolutionary algorithm problem are pre-fixed and remain static without change during optimisation processes. For example, population sizes can be initially fixed to be a value. Over the past decade considerable effort have been made to discover a general set of parameter values. Here "general" means the targeted set of parameter values can be applied to a wide range of applications across different tasks.

2.3.1.1 Tune by Hand

In earliest work evolutionary algorithm parameters are manually tuned, which to some extent is still true for many cases in contemporary applications. Historically mutation operators are mostly discussed in parameter tuning. Some famous work in this area was done by De Jong, in his doctoral thesis [31], he recommended the probability of mutation $p_m = 0.01$ and the probability of crossover $p_c = 0.6$. In other literature, Schaffer and other researchers [32] recommended $0.005 <= p_m <= 0.01$, and Grefenstette [33] recommended $p_m = 0/01$, While in [34] $p_m = 1/l$ was suggested where l denotes the the representation's length. Each set of values proposed in different papers has its own arguments which seems to help achieve optimal results in their targeted problems, where different representation and fitness function are being used.

2.3.1.2 Design of Experiments

In modern days, systematic design of experiments has been conducted in order to find optimal sets of parameters in evolutionary algorithm tasks. A comprehensive introduction to the experimental design of evolutionary computation was given in [35]. Such experiments follow the pattern of statistical experiments in terms of the inspection of parameters.

An example of this can be illustrated that, if a evolutionary algorithm is applied in a problem, then parameters involved in this program such as mutation rate are defined as factors, results or of the program such as fitness values at a prefixed generation or performance indicators such as convergence speed are defined as response, which act as a indicator as to whether the given parameters (factors) will result in better performance

in this problem. Typically in one set of experiments, all but one parameters would be kept fixed, so the resultant response is a direct reflection of this specific choice of factor. A well developed parameter tuning method called Sequential Parameter Optimization (SPO) was described in [36, 37] and has successfully been applied in many applications. Preuss and his colleagues [38] further extended SPO to be used for self-adaptation for binary representation evolutionary algorithm.

2.3.1.3 Meta-evolution

Another category of algorithms under the parameter tuning branch in our taxonomy is the meta-evolutionary algorithms, also know as nested evolutionary algorithms. In essence, meta-evolutionary algorithms include 2 levels of parameter optimisation [39], where the outer algorithm tunes the parameters of an embedded/nested inner algorithm, and the inner algorithm is responsible the optimisation of the objective function of the whole problem. Such algorithms are capable of tuning the parameters of evolutionary algorithms but can be efficient, since the outer algorithm have to be tuned first then the inner algorithms follows. An *isolation time* is defined to represent how much time the inner algorithms is allowed to optimise the objective function.

Meta-evolutionary algorithms was early investigated in [33], where the parameters of a classical genetic algorithm as the outer algorithm which was hence used for other problems. Coello [40] uses meta-evolutionary algorithm for determining the parameters of penalty function in a constrained optimisation problem. Later a method called relevance estimation and value calibration (REVAC) was proposed in [41, 42, 43] for estimation of the outcome obtained by selecting different parameters. They start with a uniform joint probability density distribution over possible parameter vectors and examine the expected performance of the evolutionary algorithm when applying new parameter vectors chosen from the distribution. And the distribution is iteratively updated according to the performance so vectors which results in better performance are more likely to be chosen over new iterations.

2.3.2 Parameter Control

In contrast with static parameters, changing parameters while evolutionary algorithms are running is more flexible and reasonable, especially if the fitness landscape changes during the optimisation process. This is called parameter control.

2.3.2.1 Deterministic Parameter Control

Earliest parameter control is deterministic, which means that the parameters are changing according the number of generations t while an evolutionary algorithm is running and searching its solutions. Under this consideration it may be a good practice to change the mutation rate during the search. Since desirably over the course of the search the algorithm will gradually concentrate its searching region more around the optimal solution, so it would be reasonable to constrain the mutation rate. [44] proposed that the mutation rate:

$$p_m(t) = \frac{1}{240} + \frac{11.375}{t^2} \tag{2.4}$$

where t denotes the number of generation. Since in the right hand side of the equation t appears in the denominator, it will make mutation rate p_m gradually decrease as the search of the algorithm carries on. Later more flexible schemes were proposed, such as [45], however they used 3 constants which must be determined with respect to their specific target problem. An updated and more successful deterministic approach was given in [46]:

$$p_m(t) = (2 + \frac{1-2}{T-1} \cdot t) \tag{2.5}$$

where the T denotes the overall number of generations of the running of the evolutionary algorithm.

Similar approach is also proposed for problems with constraint where dynamic penalty functions naturally find their applications. Typical uses are increasing dynamic penalties as the number of generation grows so as to reduce the searching region to more centre on feasible solutions. An example of such approach is proposed in [47]:

$$\tilde{f}(\mathbf{x}) := f(\mathbf{x}) + (C \cdot t)^{\alpha} \cdot G(\mathbf{x}) \tag{2.6}$$

where $f(\mathbf{x})$ is the fitness function of an individual $\mathbf{x}$ in a population and $G(\mathbf{x})$ is the constraint violation measure function while C and α are parameters.

2.3.2.2 Adaptive Parameter Control

While deterministic parameter control use a predefined deterministic equation, adaptive parameter control allows evolutionary algorithms to update parameters according to a

set of heuristic rules. An example of adaptive control is given in [14] in which the mutation strength is updated by a 1/5 success rule. The mutation steps *sigma* used in their approach will be increased if the ration of successful candidate solution is higher than 1/5 in order to make progress faster, and the steps to be decreased if the success ratio is below 1/5. This approach is not dependent on a deterministic predefined equation, but rather adapting its value according to the characteristics of the actual problems such as fitness landscape.

2.3.2.3 Self-adaptation

If no knowledge as to how to define the set of heuristic parameter updating rules is known in advance, then an evolutionary algorithm should be able to search and find the parameters settings itself. In the taxonomy we are using this is called self adaptation and will be discussed in next section.

2.4 Self-adaptive Parameter Control

Approaches belong to self-adaptive parameter control are reviewed in this section, with emphasis on self-adaptation of mutation operator, which is most related to this thesis.

2.4.1 Self-adaptive Evolutionary Algorithm

Early self-adaptation parameter control in evolutionary algorithms can be dated back to 1974, when Schwefel [48] introduced it into evolutionary strategy. Later self-adaptation was introduced into evolutionary programming [49]. It seems quite common that self-adaptation of mutation parameters with continuous representations while for binary representation evolutionary algorithms there seems no established standard in terms of how to applying self-adaptation. But some instances of attempts exists such as [50, 51] and [52].

The cumulative path-length control, a de-randomized approach to self-adaptation of evolutionary parameters, was introduced by Ostermeier [53], and gave rise to certain other algorithmic variants, e.g., the cumulative step-size adaptation (CSA) [54] and

the covariance matrix adaptation evolution strategy (CMA-ES) [55]. The latter has stood as the state-of-the-art evolutionary algorithm for many years and many successful applications can be found in literature such as in [56] [57]. Later a self-adaptive variant of the latter was given in [58].

The covariance matrix adaptation approaches such as CMA-ES computes the covariance matrix of difference of the best solutions in current generation and its parental generation. The CMSA-ES algorithm as described in [58] uses an algorithm as follows: let $\mathbf{a} = (\mathbf{x}, \sigma)$ be an individual in the population of an evolutionary algorithm, where $\mathbf{x}$ denotes the solution and σ the self-adaptive step size which is the mutation strength parameter concerned in this case. Firstly after initialisation, the algorithm uses the following steps to produce λ candidate solutions:

1. Firstly, using a global self-adaptive step size σ to produce log-normally distributed step size for each individual:

$$\sigma_i = \hat{\sigma} \exp(\tau N_i(0,1)) \tag{2.7}$$

   where τ is the learning parameter, $N_i(0,1)$ is a random value from the Gaussian distribution and the global step size $\hat{\lambda}$ is obtained by computing the average value of the step sizes from the μ best parental solutions:

$$\hat{\tau} = \frac{1}{\mu} \sum_{j=1}^{\mu} \sigma_j \tag{2.8}$$

2. Then correlated directions $\mathbf{S_i}$ are computed randomly for each individual by using a covariance matrix $\mathbf{C}$ as follows:

$$\mathbf{s_i} = \sqrt{\mathbf{C}} N_i(\mathbf{0}, \mathbf{1}) \tag{2.9}$$

   Then the random direction is scaled with the self-adaptive step size σ_i, and added to global parent solution $\mathbf{y}$ to generate

$$\mathbf{y_i} = \mathbf{y} + \sigma_i \mathbf{s}_i. \tag{2.10}$$

3. The above steps are repeated λ times, and hence the λ candidate solutions are obtained for this generation. Now the algorithm can calculate μ best solutions according to their fitness value.

4. After recombination The covariance matrix of the random directions of this generation will be generated, which is

$$\mathbf{S} = \frac{1}{\mu}\mathbf{s}\mathbf{s}^T \tag{2.11}$$

With the help of a balance parameter τ_c, the new covariance matrix of difference of the parental and current generations is computed by linear combination:

$$\mathbf{C} = (1 - \frac{1}{\tau_c})\mathbf{C} + \frac{1}{\tau_c}\mathbf{S} \tag{2.12}$$

In [58] it is suggested that

$$\tau_c = \frac{N(N+1)}{2\mu} \tag{2.13}$$

5. The algorithm terminates when the iteration of the above steps meets predefined termination conditions. The CMSA-ES algorithm is a typical case of self-adaptive parameter control in a sense that it updates its mutation strength parameter σ with the help of the covariance matrix of the difference of 2 adjacent generations.

2.4.2 Mutation Strengths

Most significant success of self-adaptation of parameters come from adapting mutation strengths, largely because mutation strength is arguably the most influential parameter in terms of the change in solution of a evolution optimisation algorithm. Increased mutation strength allows the optimisation process to search in larger solution spaces, while small mutation strength make the algorithm focus more on current solution generations.

2.4.2.1 Real-valued Mutation Strength

The above mentioned CMSA-ES algorithm is one of the example to demonstrate how the continuous mutation strength parameters are updated during the process of an self-adaptive evolutionary search. Experiments have shown such algorithms are able to change the mutation strength as the fitness landscape is changing during the evolutionary search [58][59].

An introduction to the step size σ in real-valued evolutionary algorithm is given in [60]. Other continuous mutation parameters can also be self-adapted. Self-adaptation of the skewness of the mutation distribution is given in [61] [62]. Such asymmetric mutation shows advantages in certain problem although classical approaches often assume the mutation operators is unbiased [60]. Another biased mutation is proposed in [63], where the mutation ellipsoids is self-adapted by shifting with a bias vector and shows improved performance in problem with constraint. A correlated mutation operator is proposed in [64], in which the axes of the Gaussian mutation distribution is rotated self-adaptively according the changing fitness landscape. This approach is similar to the above mentioned CMSA-ES in a sense that the adapted covariance matrix in CMSA-ES also rotate the axes of the mutation parameter distribution.

2.4.2.2 Discrete Mutation Strengths

There seems rather limited approaches for self-adaptation in evolutionary algorithms in discrete solution spaces. There are certain approaches with regard to crossover controlling for combinatoric problems using evolutionary algorithms have been proposed. Using binary string to represent crossover points for parameter control was introduced by Schaffer and Morishima [65]. A similar approach was proposed by Spears [66]. Another approach making use of integer value represented crossover points was introduced by Kramer [67]. A comparison of performance of self-adaptive inversion mutation and static inversion mutation for *GR666*, a library of travelling salesman problem, is conducted in [68].

2.4.3 Crossover Parameters

Though as an standard operator in evolutionary algorithms, crossover have not been investigated thoroughly enough compared mutation operator. One can argue it brings in diversity in searching space while others believe it simply maintains and exploits existing solutions. For example in [69, 70], it is assumed that crossover recombines different building blocks of parental generation while in [71, 60] assumes common blocks of parents are mixed. Attempts to finding self-adaptive schemes for crossover operators have been made in the project. However, approaches in this thesis, evolutionary programming for function optimisation and immune algorithm for protein folding, do not take significant advantages of crossover operator, the parameter control of crossover will not be covered in details in this section.

2.4.3.1 Crossover Probabilities

Just as the name implies, crossover probability p_c controls how often the crossover is applied. It seems not many approaches concerns the self-adaptation of crossover probabilities as it is usually set as a fixed value. For example in evolutionary strategy, the crossover operator is always applied for every solution. But in [72] it is argued that the crossover probability should be adapted during the process of evolutionary optimisation. Later an approach to self-adaptation of the crossover probability is proposed and results in several combinatorial optimisation problems was successfully reported [73]. As with the mutation strengths of mutation operators, experiments shows that self-adaptive process of an evolutionary algorithm will decrease the crossover probabilities over the course of evolutionary search. The interpretation of this might be that crossover tend to contribute in the finding of optimal solution in early stage of the search while in later stage big changes in solution spaces are not desirable.

2.4.3.2 Crossover points

Consider a solution in an evolutionary algorithms is represented as a vector string, then the crossover point is the place within the string where the solution is split into a left part and right part. Then crossover operation is done by recombining the left part and right part of two parental solutions. Instead of choosing a fixed point of the parents to be the crossover point, there have been efforts on self-adaptation of the place of crossover points. The first approach is punctuated crossover, which outperform classic genetic algorithm on 4 bench mark functions [65]. Smith and Fogarty Proposed the linkage evolving genetic operator (LEGO) algorithm, which makes use of the analogy from biology that closely located genes are more likely to be recombined. Similar approach in learning linkage was also investigated in [74] and techniques based on probabilistic modelling of the linkage learning was proposed [75].

Consider a simple case in which crossover is only applied on 1 point of the parental solutions vector strings with N bits of elements, then parameter $\sigma \in [1, N-1]$ denotes the crossover point which determine the crossover will be applied between the σth and $(\sigma+1)$th element.

Given two parental solutions

$$\mathbf{a^1} = ((p_1^1, \ldots, p_N^1), \sigma)$$

$$\mathbf{a^2} = ((p_1^2, \ldots, p_N^2), \sigma)$$

and σ as the accompanying crossover point parameters. Then initially σ are randomly chosen, then over the course of evolutionary optimisation, the algorithm will update the σ to be the new point $\sigma*$ where higher fitness value can be obtained after crossover two parents and produce the following two offspring:

$$\mathbf{a^1} = ((p_1^1, \ldots, p_\sigma^1, p_{\sigma+1}^2, \ldots, p_N^2)\sigma*),$$

$$\mathbf{a^2} = ((p_1^2, \ldots, p_\sigma^2, p_{\sigma+1}^1, \ldots, p_N^1)\sigma*),$$

2.4.4 Global Parameters

Apart from mutation parameters and crossover parameters, evolutionary algorithms also involve global parameters such as selection pressure and population size. Parameters such as mutation strengths and crossover points are local parameters which belong to individuals in a solution population. But such local information can be added together to produce global parameters. Such an approach to self-adaptation of global parameters is proposed in [76]. In the aforementioned CMSA-ES algorithm, the learning rate τ_c is just another example of self-adaptive parameter since it is computed by vote of individuals:

$$\tau_c = \frac{1}{\mu} \sum_{i=1}^{\mu} \sigma_i o \tag{2.14}$$

This clearly shows the learning rate τ_c is an aggregation of self-adaptive parameters of individuals. In [59] the self-adaptation of population size μ, λ of CMSA-ES as well as the learning rate τ_c is examined and tested, and they argue that self-adaptation of global parameters may help the algorithm search recover from a bad initialisation but there is no definitive conclusion as to whether it is overall beneficial subject to the nature of targeted optimisation problems.

2.5 Hyper-heuristic Algorithm

The word 'heuristic' is a term used to demonstrate a whole search algorithm. It also refer to a particular decision process sitting within some repetitive control structure in some circumstances. Before the proposal of "No Free Lunch Theorem" [4], some researchers have tried to argue for the absolute superiority of one heuristic over another. But later on researchers convinced that when averaged over all problems are defined on a finite search space, all search algorithms had the same average performance. The result can be seen as an intuitively natural result. The reason is that the majority of problems have no exploitable structure. Only a complete lookup table can be used in the process of defining. With the theory of "No Free Lunch Theorem", the question of what sorts of problems any given algorithm might be particularly useful will be continuously noticed.

2.5.1 Motives of Hyper-heuristic

Although researches have put forward the heuristic search methods in real-world computational search, there are still difficulties in its application. The difficulties are mainly about the significant range of parameter or algorithm choices involved. Moreover, people are lack of guidance in selecting type of approaches. The goal of hyper-heuristics is to automate the design and tune of heuristic methods, which can be used to solve hard computational search problems. The hyper-heuristics develops more applicable algorithms, which are better than many of the current implementations. It can produces generic methods based on low-level heuristics, which are much easier to implement. A hyper-heuristic can automatically produces an intelligent combination of the provided components.

The term hyper-heuristics was put forward in the early 2000s [77]. There are two fundamental ideas of hyper-heuristics. Firstly, selecting and designing efficient hybrid and cooperative heuristics is a computational search problem. Secondly, the search methodologies still need to be improved by the incorporation of learning mechanisms. More recently, the research on hyper-heuristics are mainly focus on generating heuristics automatically, which are suited to the given problems.

The ideas hyper-heuristics are derived from the early 1960s [78]. The pioneering work in 1960s proposed a method of combining scheduling rules using 'probabilistic

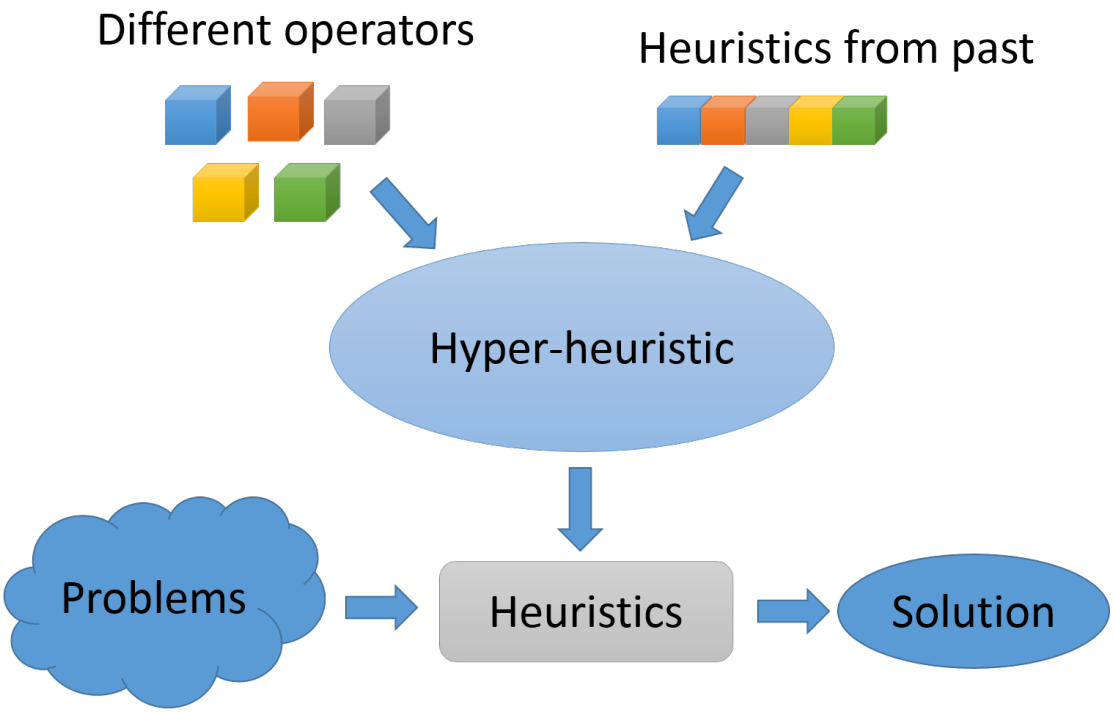

Figure 2.3: Process of hyper-heuristics

learning'. After three decades' development, hyper-heuristics were widely used. To design a good combination of problem-specific (fast) heuristics is a research problem. Researches have produced a subset of solutions in this new research by perturbing the heuristic combination and the problem data [79, 80]. In the context of the open-shop scheduling, the space of sequences of heuristic choices was researched by using a genetic algorithm. In other experiments of 1990s, researches have solved a real-world scheduling problem by using a genetic algorithm approach. Norenkov and Goodman [81] conduct a set of experiments using evolutionary algorithms. The solutions obtained were strongly affected by the subset of heuristics used.

2.5.2 Classification of Hyper-heuristics

The classification of hyper-heuristics is based on two dimensions: the nature of the heuristic search space and the source of feedback during learning. There are several types of hyper-heuristics. In this part, three approaches will be discussed: Heuristics to choose heuristics based on constructive heuristics, heuristics to choose heuristics based on perturbative heuristics and heuristics to generate heuristics.

2.5.2.1 Heuristics to choose heuristics based on constructive heuristics

These approaches are able to select and use constructive intelligently. They start with an empty solution, and build a complete solution gradually. The hyper-heuristic framework are built up with a set of pre-existing constructive heuristics. The challenge is to select the most suitable heuristic for the current problem state. When a complete solution is being reached, the process will come to the final state. Because of the complete solution, the sequence of heuristic choices is finite. They are determined by the size of the underlying combinatorial problem.

In the investigation of the application domain, five applications of the approaches have been put forward: Graph-colouring heuristics in timetabling, dispatching rules in production scheduling, packing heuristics in 1D packing problems, packing heuristics in 2D packing and cutting stock problems and variable ordering heuristics in constraint satisfaction. For example, Terashima-Marin et al [82] solve 2D-regular cutting stock problems by using the messy genetic algorithm hyper-heuristic. Garrido and Riff [83, 84] also propose a genetic algorithm hyper-heuristic to solve the 2-D strip packing problems.

2.5.2.2 Heuristics to choose heuristics based on perturbative heuristics

The search of a perturbative hyper-heuristic is conducted iteratively. It selects and applies a low-level heuristic or its subset to the current solutions to meet a set of stopping conditions. Recent proposed perturbative hyper-heuristics perform a single point. It processes a single candidate solution at each iteration.

In the recent years, perturbative hyper-heuristics have been applied in the combinatorial optimisation problems. The application of perturbative hyper-heuristics have been involved in a wide rage. For example, Kendall and Mohamad [85, 86] put perturbative hyper-heuristics into the practice of the channel assignment. Other applications include component placement, personnel scheduling, packing, planning, shelf space allocation, timetabling and vehicle routing problems.

2.5.2.3 Heuristics to Generate Heuristics

Hyper-heuristics searches a space of heuristics constructed from components, rather than a space of complete, pre-defined, heuristics. Not only it produces a solution, but

also outputs the new heuristic that produced the solution. Genetic programming [87], an evolutionary algorithm used for generating a executable computer program from a population of potential computer program, can be considered as one of the most common methodology of automatically generated heuristics.

The purpose of automatically generated heuristics is to reuse on new unseen problems of a certain class. Generally, all heuristics generated by a hyper-heuristic are reusable. They are used in a new instance to come out a legal solution. It will perform well only if being designed with re-usability. The automated heuristic design process makes human resources and time less demanding. A generated heuristic can produce a better solution than the current human created heuristic.

The research on a variety of optimisation problems of heuristics to generate heuristics has reached promising results, which are based on human-generated heuristics. Researches have shown that evolutionary computation methods are being applied in the automatically generate heuristics [88].

Different from the methodology that operates directly on the solution space, the evolutionary heuristic generation process is computationally expensive. When results are not be required for future problems, the computationally expensive can be seen as the only disadvantages of the evolutionary heuristic generation process in the short term. The evolutionary algorithm can be directly applied to the problem space. If the output or the solution are required for future problems, the entire evolutionary algorithm must be run for the second time. But if the evolutionary process can generate a quick reusable heuristic, then only one run is needed. Under the circumstance, the heuristic will perform far more quickly than an evolutionary algorithm when obtain a comparable result on the future problems.

Although the evolutionary heuristic generation process is a long process, it is quicker than manual heuristic generation. Furthermore, humans are the only sources of the potential components of the evolved heuristics. The human created heuristics is the only inspiration of the successful sets of potential components. A research has shown that the human ingenuity cannot be totally replaced by the automatic heuristic generation.

2.5.3 Hyper-heuristic and Parameter Control

As introduced in Section 2.3.2 parameter control is a different way to view self-adaptive configuration of evolutionary algorithm. It tunes algorithm parameters on-line at exe-

cution time. It is used in the evolution strategies. Feedback from the search process is used to control the mutation step size. Later in 1999, a useful classification into adaptive and self-adaptive approaches was proposed [29]. Researches also surveyed previous work on parameter control that is applied in evolutionary algorithms.

In the research of hyper-heuristics, both online and offline approaches are valuable directions. On the one hand, to find a search algorithm requires a big search effort in offline approaches. Once the algorithm has been found, the approaches will become much cheaper and faster in their application. It can be seen as a reusable method. On the other hand, online approaches are more suitable for the newly encountered instances or problems. Heuristics provide researchers with an advantageous structure. Compared with searching directly on the underlying problem space, it is much more effective to search on a space of heuristics.

An on-line methodology called reactive search advocates the integration of sub-symbolic machine learning techniques into search heuristics for solving complex optimisation problems. The purpose of the machine learning component is to let the algorithm automatically tune its operating parameters during the search operation. In a reactive feedback scheme, the learning component increases its efficiency and efficacy. The outcome of the research has been applied in the Tabu search meta-heuristic. Battiti and Brunato [89] came up with other techniques related to reactive search such as model based search, guided local search, colony optimisation and dynamic local search.

In 2007, researchers exploit the search power of multiple neighbourhoods, introducing an adaptive mechanism is Variable Neighbourhood search (VNS) [90]. VNS is employed to systematically switch neighbourboods in a predefined sequence. The research is going to explore increasing distant neighbourhoods of the current solution.

Another way of automating the design of search techniques is the algorithm portfolio method. It is firstly proposed in 1997, which obtain different return-risk profiles in the stock market by combining different stocks. An algorithm portfolio can allocate a fraction of all CPU cycles to each of them. In this way, different algorithms can be run concurrently. Most of the algorithms are immediately stopped, but the first algorithm determines the completion time of the portfolio.

2.5.4 Hyper-heuristics and Memetic Algorithms

Another approach called adaptive memetic algorithms (MAs) [91] is also closely related to the improvement of hyper-heuristics [92]. Memetic algorithms has been contributed to the improvement of hyper-heuristics. Different from memetic algorithms, the hyper-heuristics concentrates on searching in the heuristic space. Two populations, memes and genes, are maintained in the latest search simultaneously on both spaces.

2.6 Hybrid Evolutionary Algorithms

In the research area of computer sciences and technology, hybrid algorithms has been an interesting topic in recent years [93]. Nowadays researchers are paying more attention on hybrid algorithms. In order to solve difficult problems, hybrid algorithms are being used to get more powerful tools. The algorithms following this way of hybridization have been being designed by many researchers. And they are trying to design more effective algorithms to solve complex problems. A recent research has shown that hybrid algorithms are more powerful and efficient than pure algorithms.

To solve real-world problems, doing exhaustive search is not the best idea. There are still a large space to be searched, and enumerate the search space is not an efficient approach. A feasible solution is always complex. A heuristic approach can be used to help finding an optimal solution which contributes to raise the search speed, while it is also being used for obtaining at least an acceptable quality. In the last few years, a large number of heuristics have been developed. They are derived from experimental results or the arguments based on the specific problem class.

2.6.1 Motives of Hybrid Evolutionary Algorithms

The combination of global search of evolutionary algorithms and local search or other methods can bring a lot of benefits. It improves or refines an individual solution. There are many motivations to the hybridization of evolutionary algorithms.

1. Many complicated problems can be divided into several parts, in which exact methods or excellent heuristics can be used. A combination of the most appropriate methods is applicable for different sub-problems.

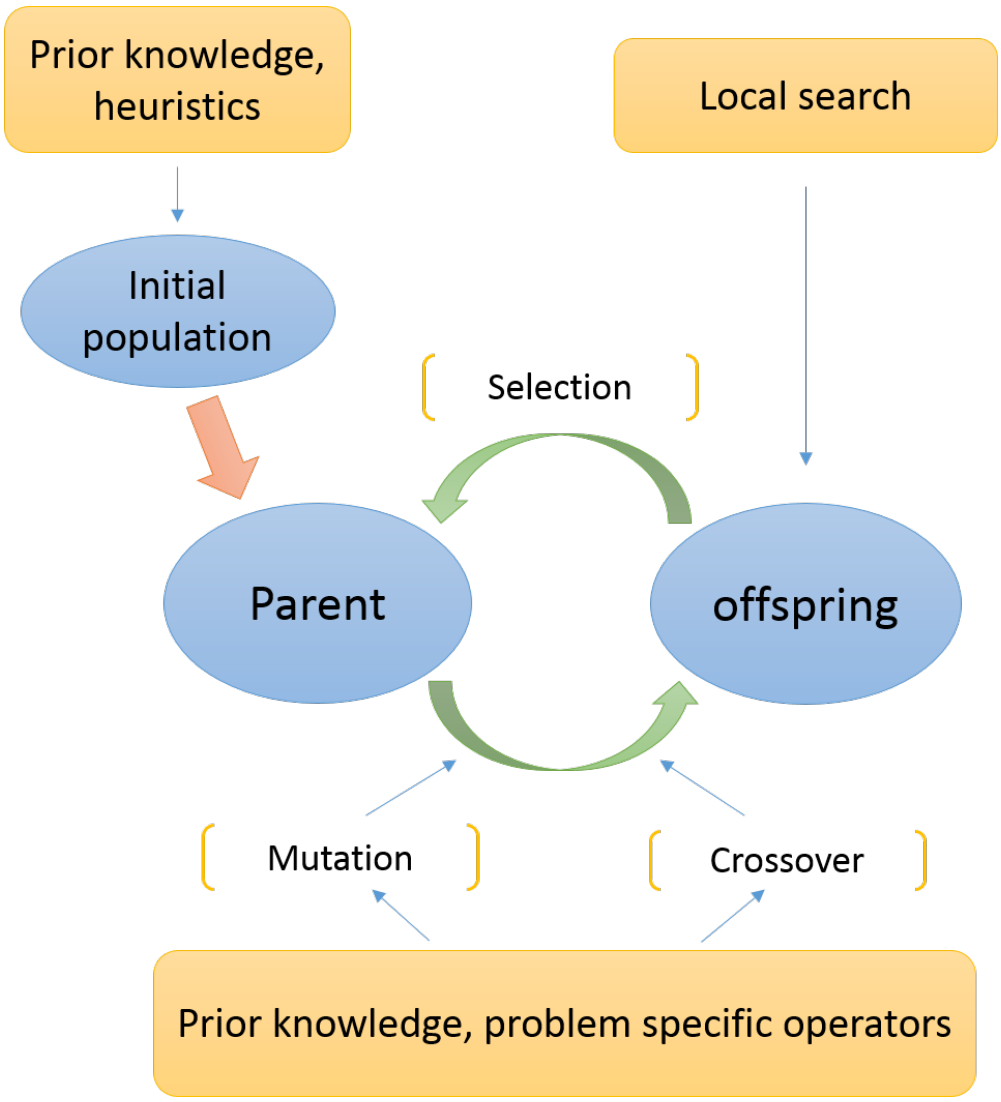

Figure 2.4: Hybridization in evolutionary algorithms

2. Successful and efficient all-purpose problem solvers do not exist. The No Free Lunch theorem [4], one of the theoretical results, has given a strong support to this view. In the perspective of Evolutionary Computing, it indicates that evolutionary algorithms are not the most excellent for global search. The competence of an evolutionary algorithm is decided by the amount of problem-specific knowledge incorporated within it.

3. The performance of evolutionary algorithms at refining near-optimal solutions are not as good as rapidly identifying good areas of the search space. By incorporating a more systematic search, EA hybrids can search good solutions in a

more efficient way. Fourthly, many problems have some constraints associated with them. Local search and other heuristics can "repair" infeasible solutions generated by standard variation operators.

2.6.2 Memetic Algorithm

Memetic Algorithms (MAs) are stochastic global search heuristics [91], which combine Evolutionary Algorithms-based approaches and local search techniques together. With the combination, the quality of the solutions created by evolution will be improved. Memetic algorithms have played an important role in many areas such as combinatorial optimization, optimization of non-stationary functions and multi-objective optimization. There are many different ways to name the methods for hybridizing EAs. For instance, hybrid genetic algorithms, Lamarckian EAs and genetic local search algorithms. Memetic algorithms covers a range of techniques which have strong relation to evolutionary-based search.

Dawkin's concept of the meme has been used as a motivation for hybridization. The application of hybridization within evolutionary algorithms could fit Dawkin's idea by using one or more phases of improvement to individual members of the population within each generation of an evolutionary algorithm.

2.6.3 Memetic Algorithm and Local Search

Local search iteratively examines the set of points in a neighbourhood of the current solution and replace the current solution with a better neighbour where possible. It is a research method. It is related to memetic algorithms. The workings of local search can be affected by three principal components.

1. The pivot rule. It gives the criteria for accepting an improving point. Both a steepest ascent and a greedy ascent pivot rule can terminate the inner loop. A steepest ascent works only after the entire neighbourhood has been searched. A greedy ascent works as soon as an improvement is found.

2. The depth of the local search. It defines the termination condition for the outer loop.

3. The neighbourhood generating function. It defines a set of points that can be reached by the application of some move operator to the point. Among the most successful global search methods, local search is the most important idea. Iterated local search makes it possible traverse a succession of "nearby" local optima. It is effective when putting into practice. Tabu Search and Simulated Annealing are the most popular heuristics based on local search. Both of them are improvement methods in the area of memetic algorithms.

2.7 Mixed Strategy in Evolutionary Algorithms

Evolutionary algorithms have been widely used and proved to be effective in a large variety of optimization domains and real-world applications. Evolutionary algorithms operate on the basis of populations, in which the objective is not only to find suitable adjustments to the current population and hence the solution, but also to perform the process efficiently.

An evolutionary algorithms is commonly designed and influenced by a set of parameters in order to provide flexibility to a specific problem [94]. Therefore, when designing an evolutionary algorithm, one should carefully choose a set of parameter for its components, which is a really time-consuming task and may fall into a optimization problem itself [29]. Furthermore, for a single problem, a parameter setting that was optimal at the beginning of a search run may become unsuitable during the evolutionary process. Thus, it is desirable to automatically modify the control parameters during the run of an evolutionary algorithms. The control of different parameters can be of various forms, ranging from mutation rates, recombination probabilities, and population size to selection operators.

In light of this, self-adaptation techniques [95, 96] have been introduced to implement such parameter control. The approach has a bias on the distribution towards appropriate directions of the search space, thereby maintaining sufficient diversity among individuals in order to enable further ability of evolution.

2.8 Mixed Strategy for Evolutionary Programming

Evolutionary programming (EP) is a branch, alongside other notable research areas such as genetic algorithms and evolution strategy, of evolutionary computation that

stems from natural biological evolution, though the differences between each branches has seen a decrease in the last two decades [12, 94]. The self-adaptive control of mutation step sizes was originally realized in the community of evolution strategy [14]. Because of its successful performance, its use gradually spread to other branches of evolutionary computation.

2.8.1 Evolutionary Programming with Self-adaptive Mutation Operators

As a key element in evolutionary programming, mutation operators have attracted significant attention in research, where the implementation of controlling mutation step sizes was further discussed on finite state machines [2], flexible molecular docking [97], as well as the optimization of numerical functions. In this project, we focus evolutionary programming on numerical function.

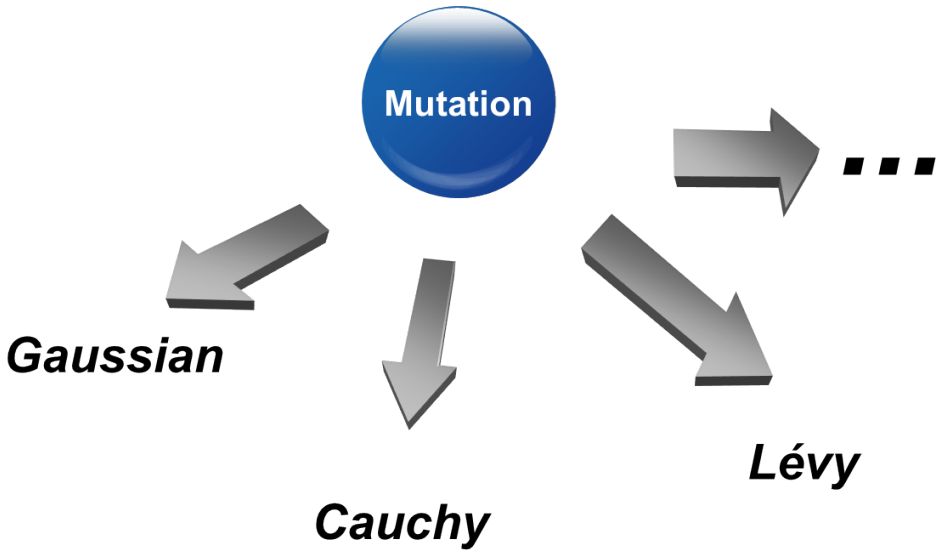

Figure 2.5: Established mutation operators available for evolutionary programming

The original mutation operator for evolutionary programming is typically Gaussian, which usually lacks robustness when applied to multi-modal functions. Therefore, con-

siderable research has been devoted to determining new mutation operator thereby preventing the search from being premature. A substitute of Gaussian mutation in fast evolutionary programming (FEP) which takes the Cauchy distribution to generate the probabilities for updating mutation operator as reported in [1], entails better performance regarding many multivariate functions. However, it is less efficient on some unimodal functions. By generalizing FEP further using mutation based on the Lévy probability distribution, further improvement can be achieved, which also establishes the relationship between the two former mutation schemes [3].

As aforementioned in the previous paragraph and shown in Fig. 2.5, there are several mutation operators which have been developed for specific problems. Unfortunately, it has been demonstrated that it is, in evolutionary algorithms, impossible to design a single algorithm or operator which always gives an efficient performance on average for a large number of problems [4], be they self-adaptation or not. The main reason is that the local fitness landscape would be changing when facing different optimization problems. More important, it can also vary at the different stage of the search process when finding the global optimal.

2.8.2 Evolutionary Programming with Mixed Strategy

An approach for enhancing the conventional EP that uses a single mutation operator is to apply different mutation operators simultaneously and integrate their advantages together. Such a strategy is called a mixed mutation strategy (borrowing the concept from game theory [6]). The employment of a mixed strategy stems from the need to explore a unified approach for maximizing the ability of various self-adaptive strategies, while assuming no prior knowledge of the problems at hand. The conception of mixed strategy came from the early work that attempted to combine different strategies together. For example, an early implementation is a linear combination of Gaussian and Cauchy distributions [5]. The drawback of this approach is that the ratio of different strategies used is short of flexibility.

An alternative approach is the improved fast EP (IFEP) [1, 3] which works by: 1) each individual of a population implementing Cauchy and Gaussian mutations simultaneously and generating two offspring; and 2) the better one being chosen to construct the next generation. Reinforcement learning theory may also be used to learn individual

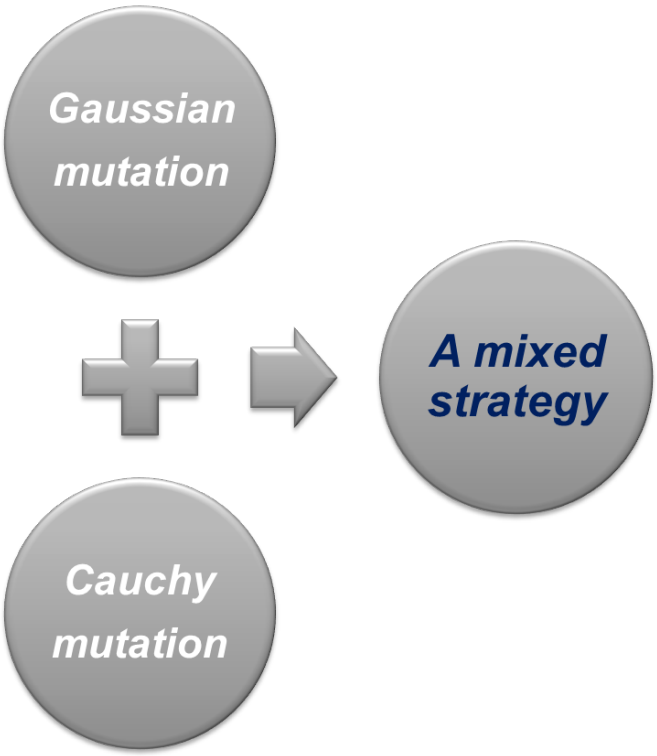

Figure 2.6: Mixed Strategy - taking advantages of both mutations

mutation operators [8]. These approaches aforementioned are simple in implementation, while all of them ignore the real-time interaction between different strategies.

Some progress has recently been made in an effort to adjust mutation strategies in evolutionary programming [6, 7], where different strategies are allowed to be used in a portion of the population in each generation of the process. Those strategies which exhibit better performance would be chosen by more individuals of the population in the next generation. The mixed strategy can thereby adapt to different optimization problem with an improved efficiency in comparison to pure strategy that uses only one single mutation operator. Since its inception, the mixed strategy has also been incorporated into other operator or even other branches of evolutionary algorithms. In [98], it is used in the crossover operator in genetic algorithms, while in [99], differential evolution is taken into account and it is also proved to be highly effective.

Figure 2.7 is an illustrative example of the process of applying the mixed strategy, in which the generations are only symbolic numbers. The general process can be described as follows:

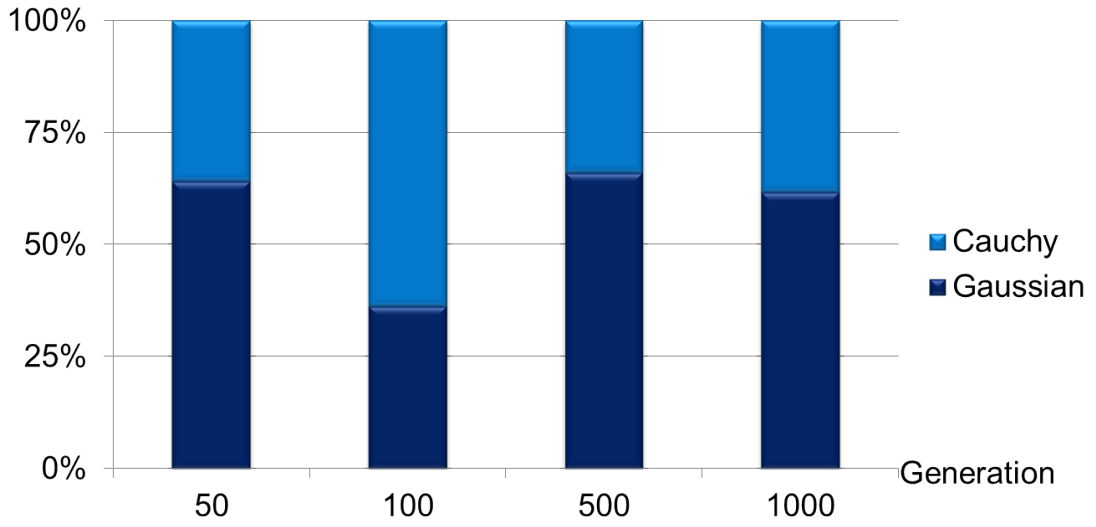

Figure 2.7: Dynamic evolution over the process of applying Mixed Strategy

- At each generation, an individual chooses one mutation operator based on a dynamic selection of probability distribution.
- This distribution will change over generations.
- At the 50th generation, Gaussian mutation may play a major role. That is, most of the individuals in the population choose Gaussian mutation and the others choose Cauchy mutation.
- However, when it comes to 100th generation, Cauchy mutation may dominate the process.
- As the number of iterations reaches 500, the probability of choosing Gaussian mutation is once again larger than the Cauchy ones.

2.8.3 A Novel Design for Mixed Strategy

In previous studies [7, 9], the design of a mixed strategy mainly utilizes the reward of each operator (i.e. an operator which produces a higher fitness will receive a better reward). However, the performance of each mutation operator is strongly linked to the fitness landscape, which could be highly dynamic at different stages of the search process. Therefore, it is important to adaptively adjust the algorithm such that the designed mixed strategy could change accordingly once the local fitness landscape changes.

In light of this, a novel mixed strategy is proposed in this project, entitled local fitness Landscape based Mixed Strategy Evolutionary Programming (LMSEP), in order for the strategy to adapt to the given fitness landscape. In this project, we give a feature to represent the ruggedness of the variety of fitness landscape. Observed from different fitness landscape, different feature value are extracted and calculated. Then a mapping is established between those value and the mixed strategy.

Two schemes for establishing the mapping are considered in this project. In the first approach, the in-line learning method, a certain mixed strategy is manually assigned to a feature value directly that is mainly on the basis of a linear mapping between strategies and features. A more advanced approach is presented afterwards, in which a training mechanism is conducted beforehand to learn the relation between features and mixed strategies. This second approach is called off-line learning in this project and employs a simplified version of K-Nearest neighbourhood method to train several training functions to automatically generate the mapping scheme.

2.9 Mixed Strategy for Clonal Selection Algorithm

A great deal of research aiming at improving the performance of CSA has been made over the last decade. Several parameters in CSA are required to define manually: antibody population size, memory pool size, selection pool size, remainder replacement size, clonal factor, number of generations, and the random number generator seed.

2.9.1 Self-adaptive Clonal Selection Algorithm

To make the search process much more automatically, an adaptive CSA (Adaptive Clonal Selection) is developed as an parameter version, which is tested on real-valued function optimization [100].

The ability of local search also attracts many research studies. For example, Lamarckian learning theory are introduced to enhance the local search of CSA in the Lamarckian Clonal Selection Algorithm in which recombination operator is also utilized to provide enough diversity for antibody population. [101, 102] Like this study, adding learning to the process are very common approaches to assist CSA. Baldwinian Clonal

Selection Algorithm is another CSA-based algorithm developed alongside a learning theory. It takes advantage of the Baldwin effect in immune system to employ information between individual (antibodies) such that the search could be better directed. It differs from Lamarckian learning theory in that the use of the exploration performed by the antigens could lead to a better guidance in the search space [103].

2.9.2 Clonal Selection Algorithm with Various Mutation Operators

Potential improvements of the existing operator in the basic algorithm are also a major focus since the inception of CSA, especially as an powerful optimization approach. Since significance of the mutation operator in CSA which does not posses a crossover operator and relies exclusively on the mutation operator to generate new antigens, a diversity of modifications of mutation strategy has been proposed.

An idea to solve complex problem is employing more mutation operators. The first thought is to implement different mutation strategies consecutively. This idea is investigated as a new approach to solve hybrid flow shop scheduling problems, in which two phased mutation procedure is implemented [104]. The generated clones undergo an inverse mutation procedure at first, then pairwise interchange mutation method is applied if the result is not favourable in the first phase. Gaussian mutation strategy, charactered with the capacity of exploitation in the local neighbourhood, is introduced in another proposed algorithm for real-valued function optimization, together with a rank based selection [105].

Cauchy mutation is used in Improved CSA (IMCSA) in order to avoid premature convergence and exhibits ability of performing fast in search of the solution for job shop scheduling problem [106]. This idea is then further extended in the Fast clonal algorithm [107] that borrows the idea from fast evolutionary programming [1], in which a parallel mutation operator comprising of Gaussian and Cauchy mutation strategy are incorporated to present an adaptive search. A chaos generator are employed to allocate both mutations aforementioned dynamically. The Cauchy mutation strategy are able to make large jumps in the search space, able to prevent the search falling into local optimum, while the Gaussian mutation shows higher probability in searching local neighbourhood, providing fine tuning ability in search of the global optima. Likewise,

another study on CLONALG for constrained optimization also consider Gaussian and Cauchy random distribution as a helpful mutation scheme that make the search more efficient [108].

Furthermore, three coding schemes are investigated in the study, including binary, gray coding as well as real-valued version. Another mutation scheme is proposed in a novel CLONALG paradigm called Artificial Immune System with Mutation Multiplicity (AISMM) where multiple mutation operators are employed simultaneously to take advantage of the information gained over a number of previous generations. The fitness gain achieved in every generation is stored and used in the selection step to determine the operator selection probabilities [109]. A CSA with binary flip mutation is implemented to solve economic load dispatch problem. The mutation rate is inversely proportional to the fitness value, with the probability of mutation varying from 0.035 to 0.010 [110].

It is also worthwhile taking into account of combining mutation and other types of operators together such that the modification of genes is in accordance with the information gained in previous generations. Like in other evolutionary algorithms, a deterministic approach was first proposed to adjust the selection of antibodies, individuals to survive and to proliferate for creating the offspring generation.

However, it is obvious that those antibodies selected by a deterministic selection operator are only those who exhibited best performance in the previous iteration, which could result in the search space falling into a relative small area and lead to a premature convergence. In light of the idea to overcome this drawback, research in CSA has been gradually turning into other thoughts of selection, especially a roulette wheel based selection mechanism. This type of selection mechanism provides helpful information in assisting the procedure of mutation while maintaining the diversity of antibodies to avoid premature convergence. A special version of roulette wheel selection in [104] is proposed in cloning process followed by a two phased mutation procedure.

Another idea is considered to apply a super mutation operator as well as a vaccination operator to modify each individual so as to maintain the diversity of antibodies [111]. Another method proposed to extend the conventional CSA is the "psychoclonal algorithm" algorithm [112] in which Maslow's need hierarchy theory is implemented in to solve the assembly-planning problem. Needs for mutation operator and are categorized into different level, alongside the employment of immune memory, and affinity

maturation, such that the solution are better guided to the global minimum rather than local ones while also preserve the ability to remove infeasible solutions.

Chapter 3

Mixed Strategy based on Local Fitness Landscape for Functional Optimisation

Every optimisation problem has its own natural characteristic, which results in the different performance and results shown by using different algorithms. In order to analyse this natural characteristic, certain approach needs to be employed that it has strong impact on the behaviour of the algorithm.

3.1 Fitness Landscape

Fitness landscape is a way of analysing various kind of situations in optimisation problems. When a meta-heuristic algorithm is running for an optimization problem, a variety of fitness are created. When those fitness are taken into account together, there are better ones and poorer ones. Among these fitness, the better ones are considered solutions and the best one is the best solution. To visualise the entirety of these fitness, the fitness landscape is constructed [113]. The solutions with better fitness are seen as peaks and those with poor fitness are valleys. Therefore, solutions are generated with those peaks and the best solution is the highest peak. The process of the search can be seen as hill climbing, where the adaptation of the algorithm can be seen as gradually moving towards the top of the hill.

3.1.1 Fitness Landscape in Biology

The idea of fitness landscape, along with genotype and phenotype, are borrowed from biology, where each genotype encodes a phenotype, and the offspring represents the productiveness [113]. In computational optimisation problems, each phenotype is assigned a fitness value according to its performance. That is, they have numerical values indicating how well the phenotype performs. Although the overall target performance is counted by the mean number of the collective of the population, each genotype is still associated with one fitness value regardless of the entire system.

If all the possible genotypes are encoded to phenotype, a fitness landscape can be generated, where the fitness of each phenotype defines the height of the landscape and each genotype defines the location with the fitness. In this way, the fitness equals to the height of the landscape, while similar genotypes are placed close to each other. In contract, those genotypes with distinct performance are placed far from each other as they would be associated with fitness values that are much more different [114].

Therefore, fitness landscapes is utilized as the idea whereby the relationship between genotypes and phenotypes can be visualised. In reality, fitness landscapes are highly dimensional and impossible to visualise. But the set of all possible genotypes, and more specifically, the degree of their similarity, and their related fitness values can still be seen as a fitness landscape.

In the concept of landscape, there are peaks and valleys, by which it can be imagined that the direction of a genotype may evolve. The genotype alters its location in a little manner in the fitness landscape every time when it performs a mutation. In the meantime, a new fitness value is assigned to its new genotype when it moves towards the next position. The higher the fitness value, the better the genotype performs, and the more likely it will create offspring which could be kept alive into the next generation. By continuing this process (search) over many generations, the genotype will eventually end up with a peak in the landscape.

3.1.2 Fitness Landscape for Meta-heuristic

Bringing the idea of fitness landscape from biology science, the ultimate target of the search is to find the highest peak of the fitness landscape, that is, the best solution,

thus solutions with more adapted condition are always preferable to less adaptive ones. However, there might be a number of local optima and a global optimum depending on the features of the landscape. It is possible that a certain search might move towards some lower peaks, from which there are no available route leading to higher peaks. In such cases, the populations get stuck in the local optima and may not move upwards to find the global one. The individuals within the population may gather around certain local optimum, wandering around this area. Therefore, further self-adaptation approaches are necessary to prevent this and help to release the population. The population would then drift down from the local peaks and start searching across the fitness landscape again.

Figure 3.1 is an example of a one dimensional function optimisation. It shows that the process of search could jump into another valley easier, or generate a different path entirely if mutations allow the genotype to make large steps across the landscape. For example, more than one point mutation in each generation are perform. This would lead to the genotype encoding with the highest possible fitness value. The shape (or view) of the entire landscape and how far mutations can move the genotype across it will determine the evolutionary direction and the final peak (global optimum) in which the genotype will going to terminate.

3.2 Local Fitness Landscape

In this section we introduce the definition of local fitness landscapes and the calculation of the number of optima. The concept of fitness landscape is one of the most commonly used metaphors to describe the behaviour of EAs in optimisation. However to give an exact definition of the concept sometimes is not easy while several different explanations exist [114].

Consider a continuous function $f(\vec{x}), \vec{x} \in \mathrm{R}^n$, where n is the dimension. The fitness landscape in a continuous space is represented by the triple (R^n, f, d), where $d(\vec{x}, \vec{y})$ is the Euclidean distance between two points $\vec{x}$ and $\vec{y}$. In a three dimensional space, it is easy to describe characteristics of a fitness landscape using intuitive words like ridges, valleys and basins etc. Nevertheless it becomes more difficult to describes the features of a fitness landscape in a higher dimensional space. Reeves [114] summarises three approaches to illustrate the features of a fitness landscape: mathematical characterisation, statistic measures and practical studies.

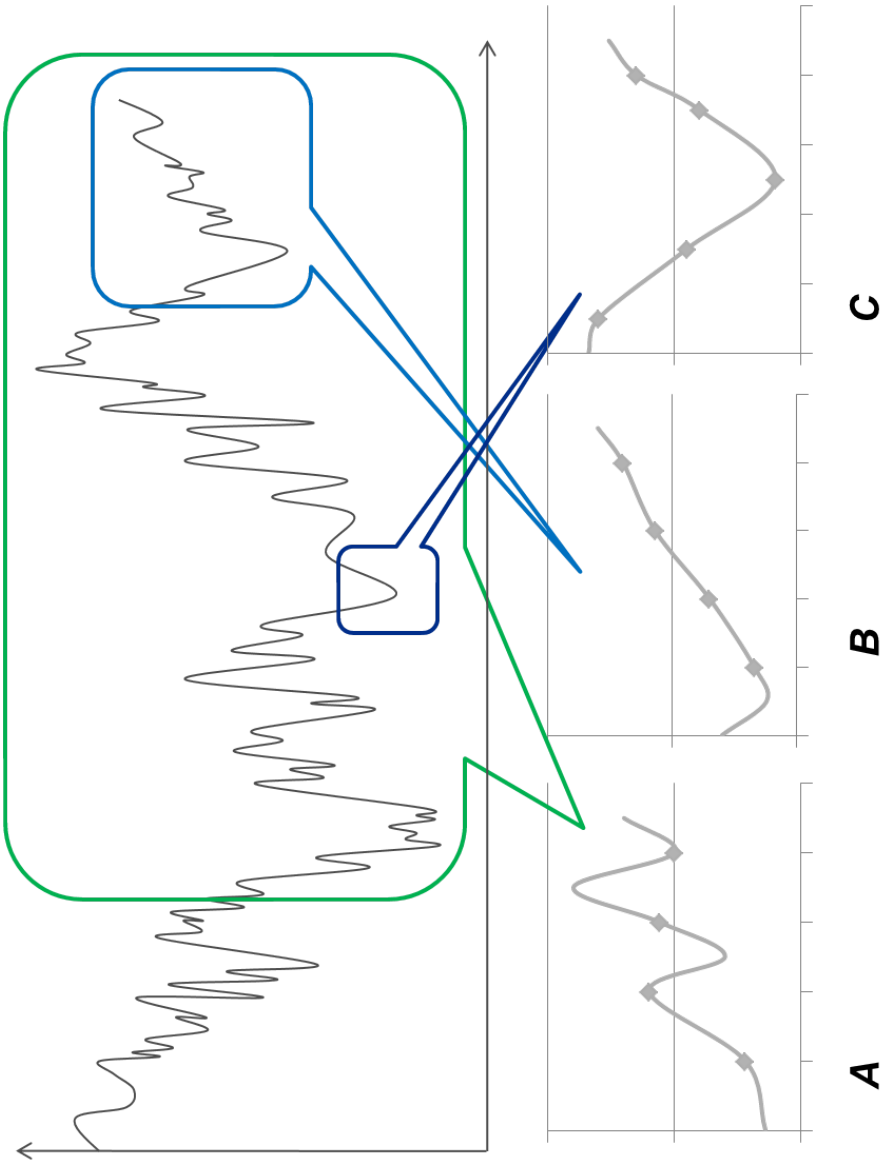

Figure 3.1: An example of fitness landscape: two dimensional function optimisation problem

We adopt a statistic measure: the number of optima on a fitness landscape (including both local and global optima). A fitness landscape with many local optima is called rugged, which intuitively means the landscape is uneven. The number of optima is strongly related to the difficulty of a fitness landscape. Usually the more number of optimal, the harder a fitness landscape. Reeves and Eremeev [115] proposed three statistical models for estimating the number of optima: waiting-time model, counting-model, and non-parametric estimation.

Either counting the exact number of optima or obtaining a statistical estimation needs a long computation time. Instead we seek a simplified approach whose computation cost is relatively low. Given a population of points $X = x(1), \cdots, x(\mu)$, a **local fitness landscape** is a part of the fitness landscape which contains the population. The purpose of introducing local fitness landscapes is that a complex fitness landscape usually consists of different local fitness landscapes such as ridges, valleys and basins. When a population resides in different areas of the fitness landscapes, its local fitness landscapes are different.

Counting the number of optima in a local fitness landscape still needs a large amount of sampling points. To simplify the computation, a new concept has been proposed [116] [117], called the observation of a local fitness landscape, which is exactly the population itself. It is represented by (X, f, d). The fitness landscape observation is only an approximation of the real fitness landscape or an observation from limited sampling points. The observation will approach the real fitness landscape as the number of points in a population increases

As can be seen in Figure 3.1, it is difficult to define whether the global fitness landscape is a unimodal or a multimodal one. However, in certain area the global fitness landscape can be seen as constructed by a great number of local ones. These local fitness landscape can be approximated by counting the population of points $X = x(1), \cdots, x(\mu)$ within a predefined window. In Figure 3.1, if individuals are distributed within Window A, then the local fitness landscape can be treated as multimodal. Similarly, different fitness landscapes also have their counterparts. When those points are located in Window B or C, the local fitness landscape is characteristic as a unimodal one.

3.3 Mixed strategy adapting to local fitness landscape

In previous studies [7, 9], the design of a mixed strategy mainly utilizes the reward of each operator (i.e. an operator which produces a higher fitness will receive a better reward). Little existing work is directly relevant to the information of local fitness landscapes. However, the performance of each mutation operator is strongly linked to the fitness landscape, so it is important to deal with the local fitness landscape where an population is located. To deal with this drawback, this project is firstly devoted to propose a novel mixed strategy in order for the strategy to adapt to the given fitness landscape.

This section describes a novel mixed strategy that is developed in order to improve the conventional evolutionary programming by two major techniques:

1. A mixed mutation strategy, based on the observation that local fitness landscapes form a key factor in the determination of the behaviour of mutations in evolutionary programming.
2. A training procedure, where several typical learning functions are introduced (as the training dataset) in order to determine the preferable probability distribution of mixed mutation operators with respect to different types of local fitness landscape.

These two tasks will be addressed below, which are then combined to deal with a set of target functions.

In evolutionary programming, a mutation operator is determined by the random variable X_j given in Eq. (2.2), which satisfies the probability distribution function F_s. A mutation operator is denoted by s. Currently, the set of mutation operators consists of Cauchy, Gaussian, Lévy and other probability distributions, and the set is denoted by $S = \{s_1, \cdots, s_L\}$.

With this notion in mind, a mixed strategy based on the probability distribution can be developed. The mixed strategy can be described as follows: at each generation, an individual chooses one mutation operator s from its strategy set based on a selection probability distribution $\rho(s)$. A mixed strategy distribution is determined by $\pi = (\rho(s_1), \cdots, \rho(s_L))$.

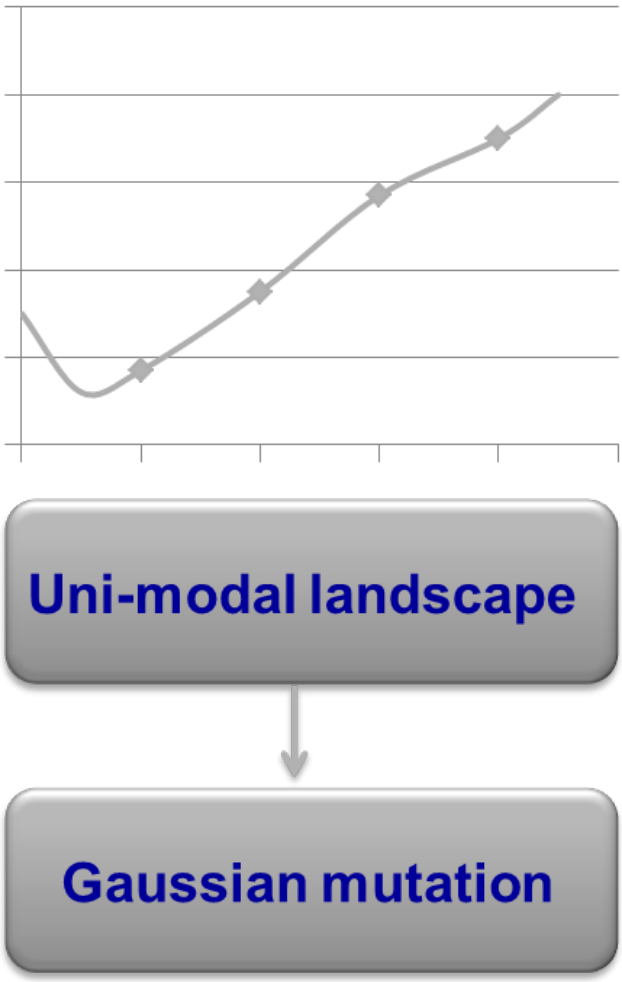

Figure 3.2: Relation between unimodal landscape and gaussian mutation

The key problem in the mixed strategy is to find out a good, if possible an optimal, probability distribution $(\rho(s_1),\cdots,\rho(s_L))$ for every individual. This distribution is dynamic, which changes over generations. The problem can be formalized as follows: Given the t-th generation population, decide a probability distribution of $\pi=(\rho(s_1),\cdots,\rho(s_L))$ which maximizes the drift towards the global optima, i.e.,

$$\max_{\pi}\{d(\vec{x}^{(t)},\vec{y});\ \vec{y}\in S_{\min})\}, \tag{3.1}$$

where $S_{\min}$ is the global optimal set, and $d(\vec{x},\vec{y})$ is the Euclidean distance.

In theory, such an optimal mixed strategy π always exists, but in practice it is impossible to find out the optimal strategy π since the optimal set $S_{\min}$ is unknown.

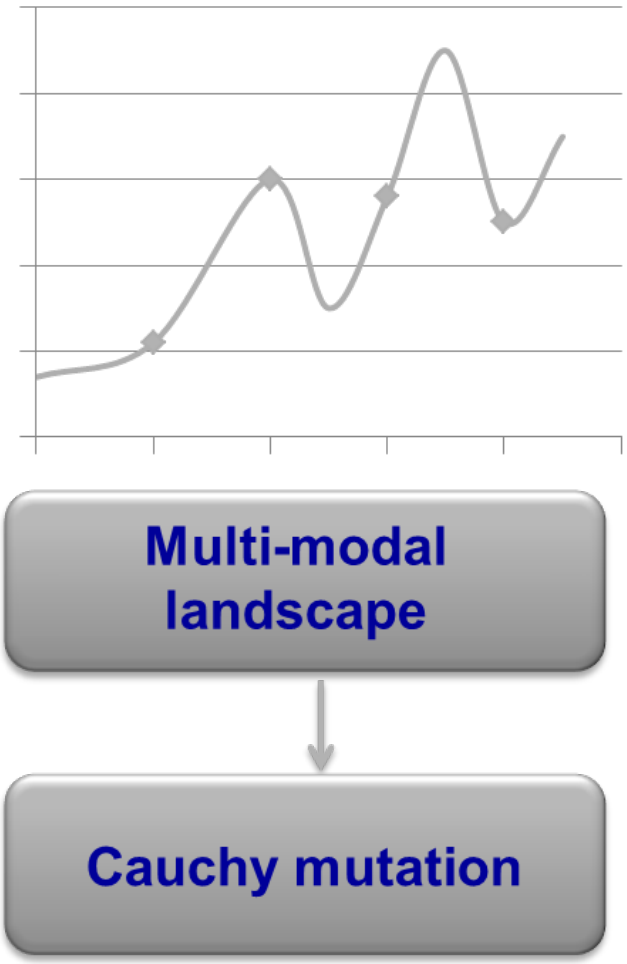

Figure 3.3: Relation between multimodal landscape and cauchy mutation

Instead, the mixed strategy is designed on the basis of following assumption that the mixed strategy should adapt to local fitness landscape. In this chapter the following principle is taken from previous experiments [6]: 1) If the local fitness landscape looks like uni-modal landscape, Gaussian mutation should be applied with a higher probability; 2) if the local fitness landscape looks like a multi-modal fitness landscape, then Cauchy mutation is applied with a higher probability.

3.3.1 Creating the Feature for Illustrating Local Fitness Landscape

There are two tasks in designing the above mixed strategy: (1) given an individual $\vec{x}$ in the real space $\mathbb{R}^n$, determine what type of local fitness landscape it looks like; (2) based

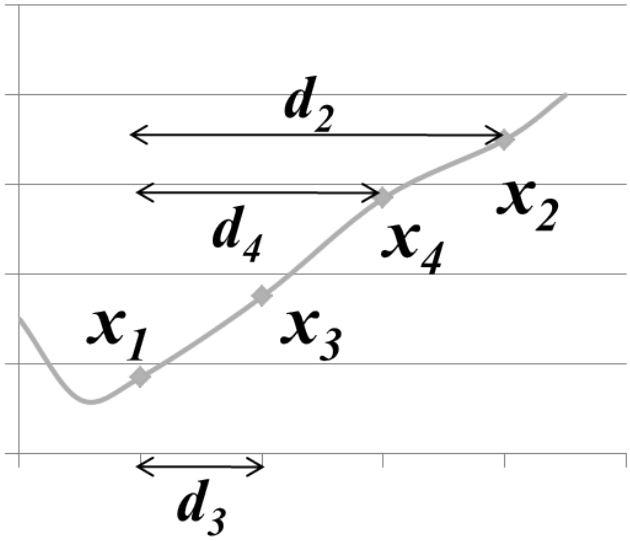

Figure 3.4: Calculate the euclidean distance between best individuals and others

on the characteristics of local fitness landscape, assign a probability distribution for the mixed strategy.

3.3.1.1 Defining the Feature Value of Local Fitness Landscapes

Consider the first task. Given an individual $\vec{x}$, it is difficult to give the precise characteristics of local fitness landscape and the computation cost will be very heavy. Instead it will be better to seek a simplified approach. Since each individual is among a population, the population forms an observation of the local fitness landscape. A simple feature of the local fitness landscape then is drawn from the observation. Sorting other individuals in the population based on their distances from the best individual in the population, then check how the fitness of each changes over the distance. If the fitness is increasing with the distance, then the local fitness landscape is like a uni-modal landscape; otherwise, it belongs to a multi-modal landscape. A simple procedure to implement this is given as follows. For a population $(x_1, \cdots, x_\mu)$,

1. Find out the best individual among the population, as shown on Fig. 3.4, mark it with x_{best}. Then calculate the distance between each individual x_i $(i = 1, \cdots, \mu)$

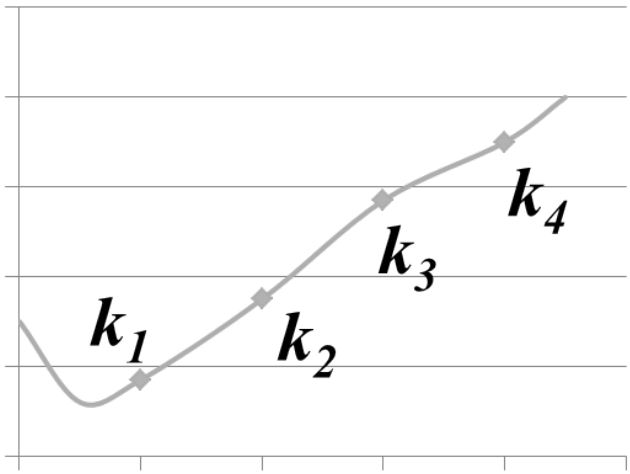

Figure 3.5: Mark the individuals with $k_1, \cdots, k_\mu$

and x_{best} as follows:

$$d_i = \sqrt{\sum_{j=1}^{n} (x_{i_j} - x_{best_j})^2}. \tag{3.2}$$

2. Sort the individuals based on the distance value, as shown on Fig. 3.5, resulting in the following in ascending order:

$$k_1, \cdots, k_\mu \tag{3.3}$$

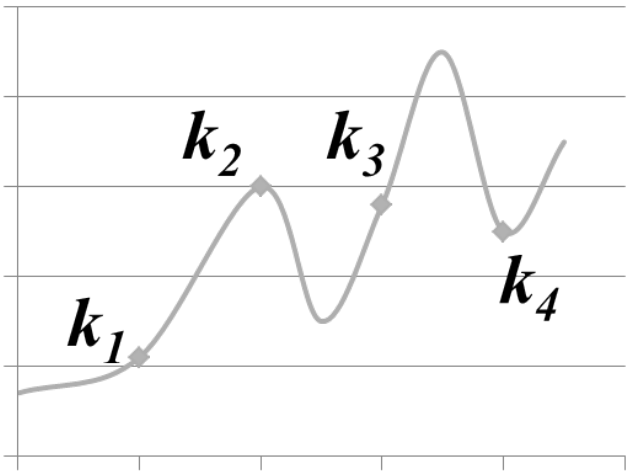

Figure 3.6: Sort the individuals based on the calculated distance (multimodal)

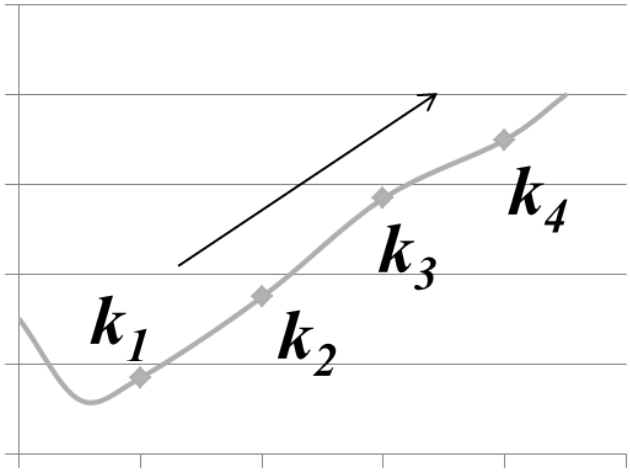

Figure 3.7: Sort the individuals based on the calculated distance (unimodal)

3. Calculate the measure of the individual on the local fitness landscape. Denote the measure value by χ. Assume the value to be 0 initially, then the value will be increased by 1 if $f_{k_{i+1}} \leq f_{k_i}$. That is, the value will be increased if there are more peaks and valleys in local fitness landscape. The value obtained from a local fitness landscape from Fig. 3.6 will be larger than the one based on Fig. 3.7.

4. Since the value got from the previous step are affected by the size of the population, it needs to be normalise as follows:

$$\varphi = \frac{\chi}{\mu}. \tag{3.4}$$

The second task is based on learning. Given several typical fitness landscapes, calculate the performance of different mixed strategy on these fitness landscapes and find the best mixed strategy for each landscape feature φ. As local fitness landscape is actual a fuzzy concept, the feature φ can be regarded as the roughness of observed fitness landscape.

So far, only two mutation operators are used, i.e., Gaussian and Cauchy mutation, though it can be easily extended to multiple mutation operators. The performance of these two mutation operators is well known [1, 6]; they behave just in an exactly opposite way. Therefore, to determine the mixed strategy $\pi = (\rho(s_{Cauchy}), \rho(s_{Gaussian}))$, a straightforward approach is used: The probability of using Cauchy mutation can be

treated to be numerically equal to the feature φ. Likewise, the probability of Gaussian mutation equals $(1-\varphi)$.

$$\begin{cases} \rho(s_{Cauchy}) & = \varphi \\ \rho(s_{Gaussian}) & = 1-\varphi \end{cases} \quad \varphi \in [0,1]. \tag{3.5}$$

Hence, for mixed strategy including only Cauchy and Gaussian mutation, the probability distribution is

$$\pi = (\varphi, (1-\varphi)), \quad \varphi \in [0,1]. \tag{3.6}$$

This is reasonable because: if the value of $\varphi = 0$, then local fitness landscape is more like a unimodal landscape, thus it is better to use Gaussian mutation only; if the value of $\varphi = 1$, then local fitness landscape is very rough, it may be good for applying Cauchy mutation only. As the value of φ increases, the probability of apply Cauchy mutations should be increased.

3.3.1.2 Procedure of New Evolutionary Programming

The details of the above mixed strategy evolutionary programming is given as follows:

1. **Initialization**: An initial population is generated consisting of μ individuals at random, each of which is represented by two real vectors $\vec{x}_i^{(0)}$ and $\vec{\sigma}_i^{(0)}$ $(i \in$

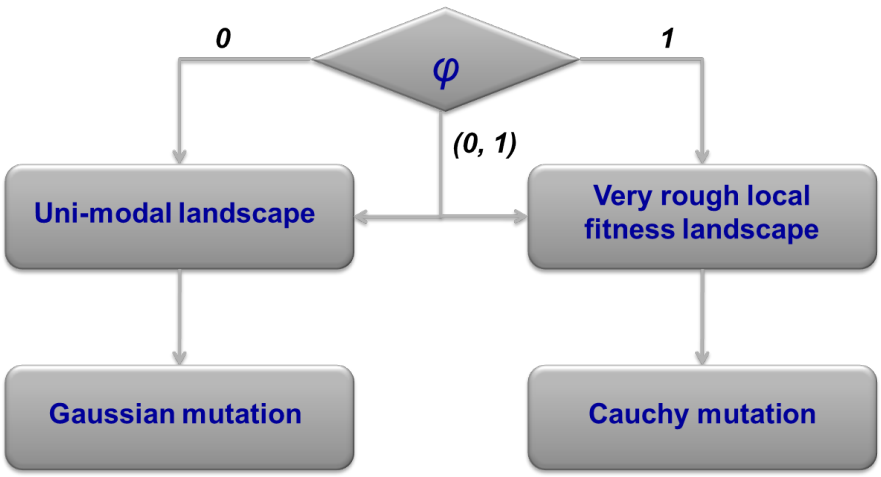

Figure 3.8: Assign the probability distribution

$1,2,\cdots,\mu$). Each $\vec{x}_i^{(0)}$ is a random point in the search space, and each $\vec{\sigma}_i^{(0)}$ is a vector of the coordinate deviations. Both vectors have n real-valued components: for $i = 1,\cdots,\mu$.

$$\vec{x}_i^{(0)} = (x_i^{(0)}(1), x_i^{(0)}(2), \cdots, x_i^{(0)}(n))$$

$$\vec{\sigma}_i^{(0)} = (\sigma_i^{(0)}(1), \sigma_i^{(0)}(2), \cdots, \sigma_i^{(0)}(n))$$

For all individuals, their mixed strategy is taken to be the same one, i.e. $\pi = (\rho^{(0)}(1), \rho^{(0)}(2),)$, where $\rho^{(0)}(1), \rho^{(0)}(2)$ represent the probabilities of choosing Gaussian and Cauchy mutation operators respectively. In the experiment, these are set to the same value initially, i.e. 0.5, to make each strategy has an equal opportunity.

2. **Mutation**: Denote t to be the generation counter. Each individual i chooses a mutation operator from its strategy set according to its mixed strategy $(\rho^{(t)}(1), \rho^{(t)}(2))$, and uses this strategy to generate a new offspring.

   The operator set includes the following two mutation operators. In each description individual parent i is written in the form $(\vec{x}_i^{(t)}, \vec{\sigma}_i^{(t)})$. The corresponding offspring i' is written in the form $(\vec{x}_{i'}^{(t)}, \vec{\sigma}_{i'}^{(t)})$.

   Gaussian mutation: Each parent i produces an offspring i' as follows: for $j = 1,2,\cdots,n$

$$\sigma_{i'}^{(t)}(j) = \sigma_i^{(t)}(j) \exp\{\tau_a N(0,1) + \tau_b N_j(0,1)\} \tag{3.7}$$

$$x_{i'}^{(t)}(j) = x_i^{(t)}(j) + \sigma_{i'}^{(t)}(j) N_j(0,1) \tag{3.8}$$

   where $N(0,1)$ stands for a standard Gaussian random variable (fixed for a given i), and $N_j(0,1)$ stands for an independent Gaussian random variable generated for each component j. The control parameter values τ_a and τ_b are chosen as follows:

$$\tau_a = 1/\sqrt{2\mu} \text{ and } \tau_b = 1/\sqrt{2\sqrt{\mu}}. \tag{3.9}$$

   Cauchy Mutation: Each parent i generates an offspring i' as follows: for $j = 1,2,\cdots,n$

$$\sigma_{i'}^{(t)}(j) = \sigma_i^{(t)}(j) \exp\{\tau_a N(0,1) + \tau_b N_j(0,1)\}, \tag{3.10}$$

$$x_{i'}^{(t)}(j) = x_i^{(t)}(j) + \sigma_{i'}^{(t)}(j) \delta_j \tag{3.11}$$

where δ_j is a standard Cauchy random variable, which is generated anew for each component j. The parameters τ_a and τ_b are set to the values used in the Gaussian mutation.

After mutation, a total of μ new individuals are generated. The offspring population is denoted by $I'(t)$.

3. **Fitness Evaluation:** Calculate the fitness of individuals in both parent and offspring populations.

4. **q-Tournament Selection**: For every individual $i \in 1,2,\cdots,2\mu$ in the parent and offspring populations, a winning function w_i is initialized to zero. For each individual i, select another individual j at random and compare fitness $f(i)$ with $f(j)$. If $f_i < f_j$, then the winning function for individual i is increased by one ($w_i = w_i + 1$). This procedure is performed q times for each individual. Based on the winning function, μ individuals from parent and offspring population with highest winning values are selected in the next generation, denoted by $I(t+1)$.

5. **Adjustment of Mixed Strategy:** For each individual i in population $I(t+1)$, its mixed strategy should be adjusted as follows: Given a population $(x_1,\cdots,x_\mu)$, assume (without losing generality) x_1 is the best individual in the population. First, calculate the feature value λ of the local fitness landscape given in Eq. (3.4). Then, adjust the mixed strategy $\rho(s)$ based on the feature value λ. Assign the probability of using Cauchy mutation is λ.

6. Steps 2-5 are repeated until the stopping criterion is satisfied.

3.3.2 Experimental Results and Analysis

The above mixed startegy EP is evaluated on 7 test functions, which were used to test IFEP in [1]. The description of these functions is given in Table 3.1. Among the selection of functions, functions f_1 and f_2 are unimodal functions, f_3 and f_4 are multimodal functions with many local minima, $f_5 - f_7$ multimodal functions with only a few local minima.

The parameter setup in the mixed EP is taken to be the same as those in [1]. Population size $\mu = 100$, tournament size $q = 10$, and initial standard deviation is taken as $\sigma = 3.0$. The stopping criterion is: to stop running at 1500 generations for functions f_1

	test functions	domain	$f_{\min}$
$f_1 =$	$\sum_{i=1}^{30} x_i^2$	$[-100,100]^{30}$	0
$f_2 =$	$\sum_{i=1}^{30} \mid x_i \mid + \prod_{i=1}^{30} \mid x_i \mid$	$[-100,100]^{30}$	0
$f_3 =$	$-20\exp\left(-0.2\sqrt{\frac{1}{30}\sum_{i=1}^{30} x_i^2}\right)$	$[-32,32]^{30}$	0
	$-\exp\left(\frac{1}{30}\sum_{i=1}^{30}\cos(2\pi x_i)\right) + 20 + e$		
$f_4 =$	$\frac{1}{4000}\sum_{i=1}^{30} x_i^2 - \prod_{i=1}^{30}\cos(x_i/\sqrt{i}) + 1$	$[-600,600]^{30}$	0
$f_5 =$	$-\sum_{i=1}^{5}\left(\sum_{j=1}^{4}(x_j - a_{ij})^2 + c_i\right)^{-1}$	$[0,10]^4$	-10.15
$f_6 =$	$-\sum_{i=1}^{7}\left(\sum_{j=1}^{4}(x_j - a_{ij})^2 + c_i\right)^{-1}$	$[0,10]^4$	-10.34
$f_7 =$	$-\sum_{i=1}^{10}\left(\sum_{j=1}^{4}(x_j - a_{ij})^2 + c_i\right)^{-1}$	$[0,10]^4$	-10.54

Table 3.1: Seven test functions, where the coefficients of $f5 - f7$ are given in

and f_3, 2000 generations for f_2 and f_4, 100 generations for $f_5 - f_7$. The lower-bound used here is $\sigma_{\min} = 10^{-5}$ for all functions except f_4. Since f_4 has a larger definition domain than the rest, $\sigma_{\min}$ is taken to be a bigger value 10^{-4}. At the initial step, the mixed strategy distribution is set to (0.5, 0.5). Results for $f_1 - f_4$ are averaged over 50 independent runs, and for $f_5 - f_7$ over 1000 independent trials.

The performance of EP using a mixed strategy mutation is compared with that of MSEP, IFEP, FEP and CEP, whose results are presented in Table 3.2 and Table 3.3. MSEP [7] is an EP with a mixed mutation strategy in which four different mutation operartors are considered and adaptively employed according to a dynamic probabilistic distribution. CEP is an EP using a Gaussian mutation and FEP stands for EP using a Cauchy mutation. In addition, the EP presented in this work which is related to the local fitness landscape is denoted by LMESP.

Table 3.2 lists the results of the mean best fitness generated in benchmark functions, from which it is obvious that LMSEP performs much better than IFEP, FEP and CEP over all test functions except on f_4. However, LMSEP still produces a reasonable result at a similar level of precision which reveals that the mixed strategy performs at least as well as a pure strategy. On $f_1 - f_3$, it is observed that LMSEP performs better or as well as that of MSEP [7]. It can be seen that, with the use of MSEP, a considerably improved result for f_4 can be obtained, which is stated in [7]. The main reason for this is that there are four types of mutation operation applied in the mixed strategy, so that

		LMSEP	MSEP [7]	IFEP [1]	FEP [7]	CEP [7]
	generations	mean best	mean best	mean best	mean best	mean best
f_1	1500	4.964e-5	1.0e-4	4.16e-5	2.3e-3	5.2e-4
f_2	2000	1.957e-3	4.1e-4	2.44e-2	8.1e-3	2.6e-3
f_3	1500	1.845e-3	1.7e-3	4.83e-3	5.2e-2	15.1
f_4	2000	5.479e-2	8.5e-4	4.54e-2	3.9e-2	8.6e-2
f_5	100	-9.074	-10.15	-6.49	-4.81	-5.54
f_6	100	-9.688	-10.4	-7.10	-5.91	-8.84
f_7	100	-9.719	-10.54	-7.80	-8.73	-9.58

Table 3.2: Comparison of mean best fitness between LMSP and MEP, IFEP, FEP, CEP

the process of the experiment has a relative higher flexibility than LMSEP presented in this section. Note that the lower bound $\sigma_{\min}$ using in the experiment is also an important factor with respect to these results.

		LMSEP	MSEP [7]	FEP [7]	CEP [7]
	generations	Std. dev.	Std. dev.	Std. dev.	Std. dev.
f_1	1500	1.43e-5	1.3e-5	2.2e-3	5.4e-4
f_2	2000	2.46e-4	2.1e-5	7.7e-4	1.7e-4
f_3	1500	1.87e-3	4.3e-4	2.5e-2	2.6
f_4	2000	3.48e-3	1.3e-3	2.3e-2	0.12
f_5	1000	2.37	5.0e-5	0.18	1.48
f_6	1000	1.92	4.7e-6	1.57	1.41
f_7	1000	2.26	1.3e-4	0.87	0.68

Table 3.3: Comparison of standard deviation between LMESP and MEP, FEP, CEP

The standard deviation of LMSEP whose evaluation is given in Table 3.3 is also compared with those obtained using other approaches. According to the results, LMSEP exhibits a better performance on $f_1 - f_4$ and a similar performance on $f_5 - f_7$ in

comparison to a pure strategy, namely FEP or CEP. This fact indicates that LMSEP has a more stable performance on unimodal problems, being comparable with a pure strategy and of the a same stability on multimodal problems. However, LMSEP does not present a better performance than MSEP. Having been illustrated previously, it is affected by the types of mutation strategy introduced in the experiment. Furthermore, the adjustment of the feature of local fitness landscape is fairly straightforward. The implementation of the parameter, λ in this section, can be modified in future research so that a better result can be expected.

3.3.3 Discussion

This section has presented a new evolutionary programming using mixed strategies, LMSEP, to combat the drawbacks of conventional EPs that employ a single mutation operator. Efforts have been made in order to explain why and how LMSEP works, which is characterized by the local fitness landscape using a mixture of different mutation strategies.

The performance of LMSEP is tested on a suite of 7 benchmark functions and compared with previously studied EPs. The experimental results confirmed that the new approach has the ability to perform at least as well as the best of different conventional strategies with single mutations. Furthermore, the tests regarding to standard deviation also demonstrated that LMSEP has a more stable performance, which helps in offering a reasonable results in potential real world applications.

Many aspects remain to be addressed in the future. The experiment will be extended to more types of problem and more complicated functions. A fine adjustment of local fitness landscape remains to be considered. As a compatible satisfactory result can be obtained using MSEP, a better implementation of λ parameter may be valuable and lead to more stable performance. Additionally, more mutation operators can be taken into account, (e.g. Lévy mutation). Furthermore, introducing mixed strategies to other types of operator like crossover and selection also forms a interesting piece of future work.

3.4 Training Mechanism for Learning Local Fitness Landscape

The above study shows the design of a novel mixed mutation strategy based on local fitness landscapes. Its implementations involves the fluctuation of the proportion for each mutation operator to be applied in every generation. As the performance of Cauchy and Gaussian mutation is well-known (with them simply behaving in an opposite way), the mixed strategy π can be determined by Eq. (3.5). However, this implies that an existing mixed strategy corresponding to a certain local fitness landscape has already been determined via human intervention. It also makes the algorithm resistant to the use of any novel mutation operator without sufficient prior knowledge about the problems at hand. In view of this, it is desirable if the mixed strategy S_x regarding to the given local fitness landscape φ_x can be self-determined and generalized to similar cases.

3.4.1 Implementation of Training Process

The training can be accomplished by introducing a training procedure prior to running the algorithm on target functions. The task of finding the global minimum in numerical function optimization is herein implemented by learning rules based on experience gained from prior performance [118]. In particular, a suite of functions are considered as the training examples. The set of training functions $\{f_1, \cdots, f_\gamma\}$ is chosen to be the representatives of different local fitness landscapes, e.g., unimodal functions, multimodal functions with many local minima, and multimodal functions with only a few local minima.

3.4.1.1 Defining the Feature Value of Local Fitness Landscapes

Features of local fitness landscapes φ, as well as the corresponding mixed strategy, are required to construct the actual training data. They can be obtained by taking the advantage of the algorithm presented in Section 3.3. Note that, as the algorithm is used to test the entire process of function optimization, it may be performed all along until certain performance criteria are satisfied or the maximum execution time is reached.

Under normal circumstances, the algorithm will terminate at a plateau state, suggesting it could not find any better result. Also, prior to reaching this state, it usually will have experienced an entire operation region involving a large number of intermediate states. These intermediate states may exhibit a variety of fitness landscapes that may vary in terms of the lapse of time. Hence, if the algorithm runs on a multimodal function, after running a large number of generations, individuals in the population may shrank to a limit region in the vicinity of the global optimal, in favour of similar results. Since differences between them will be considerably small, the underlying local fitness landscape will look like a unimodal.

In order to ensure that all individuals are located uniformly and randomly in the entire domain of training functions, the algorithm is slightly modified. This is achieved by running a relative small number t_T of generations by which the results of each training function are averaged.

According to Eq. (3.4), feature φ can be set to a different value $\{\varphi_1, \cdots, \varphi_n\}$, based on a set of training functions $\{f_1, \cdots, f_n\}$. Thus, the probability distribution of the mutation operator is needed. For a mixed strategy involving only Cauchy and Gaussian mutation, it can be calculated using Eq. (3.5). To be consistent with φ, the probability distribution is also averaged by t_T as follows:

$$\begin{cases} \rho(s_{Cauchy}) & = \sum_{i=1}^{t_T} \dfrac{\varphi_i}{t_T} \\ \rho(s_{Gaussian}) & = \sum_{i=1}^{t_T} \dfrac{1-\varphi_i}{t_T} \end{cases} \quad \varphi_i \in [0,1]. \tag{3.12}$$

While $\varphi, \cdots, \varphi_k$ are every single value obtained from every generation among t_T ones.

As γ individual functions are chosen to form the training examples, γ pairs of features and probability distribution are calculated. The set of features $\{\varphi_1, \cdots, \varphi_\gamma\}$ and probability distributions $(\rho(s_1), \cdots, \rho(s_L))$ have a one-to-one correspondence. It is reasonable to assume that a target probability distribution, given a feature, can be learned from the underlying correspondence by taking advantage of the existing ones, $\{\varphi_1, \cdots, \varphi_\gamma\}$.

In order to implement the learning of the correspondence between feature φ_x and the target, π_x, a distance-based approach is utilized to approximate the best π_x. All

training data as well as π_x are considered as points in a line in terms of the values of feature φ. Two points with a relatively low distance are regarded as neighbours. Given a target feature φ_x whose value is in the interval $[0,1]$, one instance φ_a among training data $\{\varphi_1, \cdots, \varphi_\gamma\}$ can be found as its nearest neighbour. Therefore, this approach has an intuitive appealing, treating π_a that is associated with φ_a as the required approximation. The probabilities of each mutation operator in the required π_x are

$$\rho_x(s) = \rho_a(s). \tag{3.13}$$

It is reasonable to adopt Eq. (3.13) when there is only one point in the vicinity of target feature π_x. However, it is possible that the target feature is given in the location where the distances

$$d(\varphi_x, \varphi_i) = \mid \varphi_x - \varphi_i \mid . \tag{3.14}$$

from its two neighbour value generated from training function are nearly the same. Having taken notice of this, the approximation above is modified so that the two nearest neighbours of the target feature (one on each side) are both taken into account. The contribution of each neighbour is weighted according to its distance to the query point φ_x in order to place a greater weight onto closer neighbours. Denoting the neighbour with a smaller value as φ_a and the other as φ_b, the relative placement factor is defined by

$$\lambda = \frac{\varphi_x - \varphi_a}{\varphi_b - \varphi_a}, \ \ \lambda \in [0,1].$$

The probability of each mutation operator of target π_x is then considered as the linear combination of its two neighbours:

$$\rho_x(s) = (1-\lambda)\rho_a(s) + \lambda\rho_b(s), \ \ \lambda \in [0,1].$$

Note that if the target feature φ happens to take a value close to 0 or 1, Eq. (3.13) can then be used. In this case, it is obvious that Eq. (3.4.1.1) is not applicable because there is only one existing neighbour.

In summary, the calculation of required π is carried out as follows:

Distance Rule:

1. Given a target feature φ_x, identify its two nearest points φ_a and φ_b among γ training features.

2. Examine the value of φ_a and that of φ_b relative to φ_x. If they are both larger or less than φ_x, then target distribution π is generated as stated in Eq. (3.13). Otherwise, Eq. (3.4.1.1) is adopted.

3.4.1.2 Number of Generations for Training Process

When a population of searching individuals are placed uniformly in the entire fitness landscape of a training function, the entire fitness landscape can be considered as a stereotype of certain local fitness landscape, $\Psi_g,\ g \in 1, \cdots, \gamma$. It could be of a multimodal, a unimodal, or a combination of multimodal and unimodal. The process will utilise a mixed strategy probability distribution, π_g, which is directly relevant to the entire landscape, Ψ_g.

When the population are placed in a certain local fitness landscape similar to the stereotype Ψ_g in the real test, we can assign a similar mixed strategy π_g to it. Therefore, the only thing we need to know from the training is what Ψ_g looks like, which means to obtain the value of feature φ_g for the entire fitness landscape.

However, after a large number of generations, the locations of the population will automatically concentrate in a constraint area of the landscape. λ at that stage is no longer directly affected by the entire fitness landscape Ψ_g, but affected by a part of it. Therefore, to generate the required λ, the training should be limited to the early stage of the search process.

Moreover, the value can not be too small either. Because of the random distribution of the initial population, the individuals might not be placed uniformly across the entire landscape in the first generation. We need several generations to allow the population traverse across the entire fitness landscape. As a result, the search process for each training function will be run for only 5 generations.

3.4.1.3 Procedure for Training

With the aid of aforementioned training procedure, the mixed strategy can be approximately generated automatically in relation to previously unknown local fitness landscapes. The details of this new mixed strategy-based evolutionary programming algorithm are given as follows:

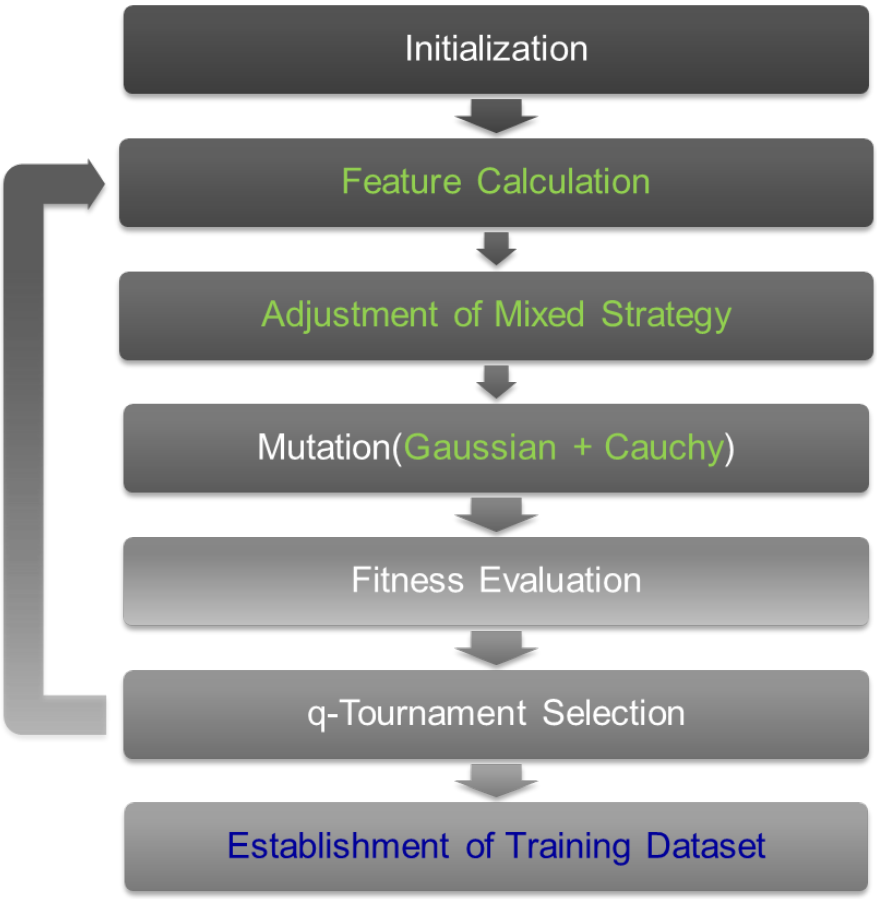

Figure 3.9: Training - only 5 generations for every function

Training: Before applying the mixed strategy, γ functions are employed as training examples so as to generate a set of correspondence between feature φ and probability distribution π of mixed strategy.

T1: **Initialization:** An initial population is generated consisting of μ individuals at random, each of which is represented by two real vectors $\vec{x}_i^{(0)}$ and $\vec{\sigma}_i^{(0)}$ $(i \in 1,2,\cdots,\mu)$. Each $\vec{x}_i^{(0)}$ is a random point in the search space, and each $\vec{\sigma}_i^{(0)}$ is a vector of the coordinate deviations. Both vectors have n real-valued components: for $i = 1,\cdots,\mu$.

$$\vec{x}_i^{(0)} = (x_i^{(0)}(1), x_i^{(0)}(2), \cdots, x_i^{(0)}(n))$$

$$\vec{\sigma}_i^{(0)} = (\sigma_i^{(0)}(1), \sigma_i^{(0)}(2), \cdots, \sigma_i^{(0)}(n))$$

For all individuals, their mixed strategy is taken to be the same one, i.e. $\pi = (\rho^{(0)}(1), \rho^{(0)}(2),)$, where $\rho^{(0)}(1), \rho^{(0)}(2)$ represent the probabilities of choosing Gaussian and Cauchy mutation operators respectively.

T2: **Feature Calculation:** For each individual i in the population, the feature of local fitness landscape can be determined as follows: given a population $(x_1, \cdots, x_\mu)$, assume x_1, without losing generality, is the best individual in the population. Calculate the feature value φ of the local fitness landscape given in Eq. (3.4).

T3: **Adjustment of Mixed Strategy:** Adjust the probability distribution π_m based on the feature value φ according to Eq. (3.5). Assign φ to the probability of using Cauchy mutation, $(1-\varphi)$ to Gaussian mutation.

T4: **Mutation:** Denote t to be the generation counter. Each individual i chooses a mutation operator from its strategy set according to its mixed strategy $(\rho^{(t)}(1), \rho^{(t)}(2))$, and uses this strategy to generate a new offspring.

The operator set includes the following two mutation operators. In each description individual parent i is written in the form $(\vec{x}_i^{(t)}, \vec{\sigma}_i^{(t)})$. The corresponding offspring i' is written in the form $(\vec{x}_{i'}^{(t)}, \vec{\sigma}_{i'}^{(t)})$.

Gaussian mutation: Each parent i produces an offspring i' as follows: for $j = 1, 2, \cdots, n$

$$\sigma_{i'}^{(t)}(j) = \sigma_i^{(t)}(j) \exp\{\tau_a N(0,1) + \tau_b N_j(0,1)\}$$

$$x_{i'}^{(t)}(j) = x_i^{(t)}(j) + \sigma_{i'}^{(t)}(j) N_j(0,1)$$

where $N(0,1)$ stands for a standard Gaussian random variable (fixed for a given i), and $N_j(0,1)$ stands for an independent Gaussian random variable generated for each component j. The control parameter values τ_a and τ_b are chosen as follows:

$$\tau_a = 1/\sqrt{2\mu} \text{ and } \tau_b = 1/\sqrt{2\sqrt{\mu}}.$$

Cauchy Mutation: Each parent i generates an offspring i' as follows: for $j = 1, 2, \cdots, n$

$$\sigma_{i'}^{(t)}(j) = \sigma_i^{(t)}(j) \exp\{\tau_a N(0,1) + \tau_b N_j(0,1)\},$$

$$x_{i'}^{(t)}(j) = x_i^{(t)}(j) + \sigma_{i'}^{(t)}(j) \delta_j$$

where δ_j is a standard Cauchy random variable, which is generated for each component j. The parameters τ_a and τ_b are set to be the values used in the Gaussian mutation.

After mutation, a total of μ new individuals are generated. The offspring population is denoted by $I'(t)$.

T5: **Fitness Evaluation:** Calculate the fitness of individuals in both parent and offspring populations.

T6: **q-Tournament Selection:** For every individual $i \in 1,2,\cdots,2\mu$ in the parent and offspring populations, a winning function w_i is initialized to zero. For each individual i, select another individual j at random and compare fitness $f(i)$ with $f(j)$. If $f_i < f_j$, then the winning function for individual i is increased by one ($w_i = w_i + 1$). This procedure is performed q times for each individual. Based on the winning function, μ individuals from parent and offspring population with highest winning values are selected in the next generation.

T7: **Establishment of Training Dataset:** Steps T2-T6 are repeated for γ times which is the stopping criterion of training procedure. Then, φ and π_m are averaged by following Eq. (3.12). Once this training dataset is established, no such learning procedures are required to be repeated in real test. This is because the testing procedures only require the use of the mixed strategies resulted from this training process. Therefore, the running cost of training procedure is excluded when evaluating the total cost of testing target functions.

3.4.1.4 Procedure for Real Test on Target Function

Upon the completion of training, the EP with the proposed mixed strategy can be run on various target functions, utilising the information from the already established training dataset:

P1: The testing procedure involves a general procedure of EP combined with Feature Calculation in preparation for the use of the mixed strategy. Several steps are to be performed in the way that are identical to what is done during the Training Procedure. In particular, the first two steps are identical to T1 and T2.

P2: **Learning Mixed Strategy:** With the knowledge of feature φ_x, the mixed strategy related to it is then determined as follows: Find two similar features in the training dataset using the distance defined in Eq. (3.4.1.1); Consider their relative positions to φ_x, the probability distribution of the mixed strategy is then obtained according to **Distance Rule**.

P3: After this, the steps are also the same to the corresponding ones (T4-T6) of the training procedure.

P4: Steps P1-P2 are repeated until the stopping criterion is satisfied.

3.4.2 Experimental Results and Analysis

For the purpose of validating the effectiveness of proposed mixed strategy based on local fitness landscape, 23 functions which were used to test FEP in [1] are employed as the training examples. The description of these functions is provided in Table 3.4. They are divided into 3 classes: functions $f_1 - f_7$ are unimodal functions, functions $f_8 - f_{13}$ are multimodal functions with many local minima, and functions $f_{14} - f_{23}$ are multimodal functions with only a few local minima. The parameter setup in the mixed EP is taken to be the same as those in [1]. Population size $\mu = 100$, tournament size $q = 10$, and initial standard deviation is taken as $\sigma = 3.0$. The lower-bound used for them is $\sigma_{\min} = 10^{-5}$ for all functions except f_8 and f_{11}. Since f_8 and f_{11} have larger definition domains than the rest, $\sigma_{\min}$ is taken to be a bigger value 10^{-4}.

To conduct the process of training, some of the functions are chosen as training samples which should consist of all the three types of functions. In this work, functions f_3, f_5, f_8, f_{13}, f_{14}, f_{17} and f_{20} are chosen as representatives of three classes. The generations t_T of training functions are limited to 5. These 7 training functions are coloured in grey background in Table 3.4.

All the other functions are used as actual testing functions. The stopping criterion is: to stop running at 1500 generations for functions f_1, f_6 f_{10} and f_{12}, 2000 generations for f_2 and f_{11}, 5000 generations for f_4 and f_9, 4000 generations for f_{15}. The rest will run 100 iterations. The lower-bound used for them is $\sigma_{\min} = 10^{-5}$ for all functions except f_4. Results for $f_1 - f_{15}$ are averaged over 50 independent runs, and for $f_{16} - f_{23}$ over 1000 independent trials.

The performance of EP using a mixed strategy mutation is compared with that of MSEP, SPMEP, LEP, FEP and CEP, whose results are presented in Table 3.6 and Table 3.7. For simplicity, the algorithm proposed in this work (which is related to the local fitness landscape) is named LMSEP hereafter. MSEP [7] is an EP with a mixed mutation strategy in which different mutation operators are considered and adaptively employed

Table 3.4: 23 Functions used for experiments(Part 1).

Test functions	n	Domain	$f_{\min}$
$f_1 = \sum_{i=1}^{30} x_i^2$	30	$[-100,100]^n$	0
$f_2 = \sum_{i=1}^{30} \mid x_i \mid + \prod_{i=1}^{30} \mid x_i \mid$	30	$[-10,10]^n$	0
$f_3 = \sum_{i=1}^{30} \left(\sum_{j=1}^{i}\right)^2$	30	$[-100,100]^n$	0
$f_4 = \max\{\mid x_i \mid, 1 \leq i \leq n\}$	30	$[-100,100]^n$	0
$f_5 = \sum_{i=1}^{29} \left[100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2\right]$	30	$[-30,30]^n$	0
$f_6 = \sum_{i=1}^{30} (\lfloor x_i + 0.5 \rfloor)^2$	30	$[-100,100]^n$	0
$f_7 = \sum_{i=1}^{30} i x_i^4 + \text{random}[0,1)$	30	$[-1.28,1.28]^n$	0
$f_8 = -\sum_{i=1}^{30} \left(x_i \sin(\sqrt{\mid x_i \mid})\right)$	30	$[-500,500]^n$	-12569.5
$f_9 = \sum_{i=1}^{30} \left[x_i^2 - 10\cos(2\pi x_i) + 10\right]$	30	$[-5.12,5.12]^n$	0
$f_{10} = -20\exp\left(-0.2\sqrt{\frac{1}{30}\sum_{i=1}^{30} x_i^2}\right) - \exp\left(\frac{1}{30}\sum_{i=1}^{30}\cos(2\pi x_i)\right) + 20 + e$	30	$[-32,32]^n$	0
$f_{11} = \frac{1}{4000}\sum_{i=1}^{30} x_i^2 - \prod_{i=1}^{30}\cos(\frac{x_i}{\sqrt{i}}) + 1$	30	$[-600,600]^n$	0
$f_{12} = \frac{\pi}{30}\left\{10\sin^2(\pi y_1) + \sum_{i=1}^{29}(y_i - 1)^2\left[1 + 10\sin^2(\pi y_{i+1})\right] + (y_n - 1)^2\right\}$ $+ \sum_{i=1}^{30} u(x_i, 10, 100, 4), \quad y_i = 1 + \frac{1}{4}(x_i + 1)$	30	$[-50,50]^n$	0
$f_{13} = 0.1\left\{10\sin^2(\pi 3x_i) + \sum_{i=1}^{29}(x_i - 1)^2\left[1 + \sin^2(3\pi x_{i+1})\right]\right.$ $\left. + (x_n - 1)^2\left[1 + \sin^2(2\pi x_{30})\right] + \sum_{i=1}^{30} u(x_i, 5, 100, 4)\right\}$	30	$[-50,50]^n$	0

Table 3.5: 23 Functions used for experiments(Part 2).

Test functions	n	Domain	$f_{\min}$
$f_{14} = \left[\frac{1}{500} + \sum_{j=1}^{25} \frac{1}{j+\sum_{i=1}^{2}(x_i - a_{ij})^6}\right]^{-1}$	2	$[-65.536, 65.536]^n$	0.998
$f_{15} = \sum_{i=1}^{11} \left[a_i - \frac{x_i(b_i^2+b_i x_2)}{b_i^2+b_i x_3+x_4}\right]^2$	4	$[-5,5]^n$	0.0003075
$f_{16} = 4x_1^2 - 2.1x_1^4 + \frac{1}{3}x_1^6 + x_1x_2 - 4x_2^2 + 4x_2^4$	2	$[-5,5]^n$	-1.0316285
$f_{17} = (x_2 - \frac{5.1}{4\pi^2}x_1^2 + \frac{5}{\pi}x_1 - 6)^2 + 10(1 - \frac{1}{8\pi})\cos x_1 + 10$	2	$[-5,10] \times [0,15]$	0.398
$f_{18} = \left[1 + (x_1+x_2+1)^2(10 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2)\right]$ $\times \left[30 + (2x_1 - 3x_2)^2 \times (18 - 32x_1 + 2x_1^2 + 48x_2 = 36x_1x_2 + 27x_2^2)\right]$	2	$[-2,2]^n$	3
$f_{19} = -\sum_{i=1}^{3} c_i \exp\left[-\sum_{j=1}^{4} a_{ij}(x_j - p_{ij})\right]$	3	$[0,1]^n$	-3.86
$f_{20} = \sum_{i=1}^{4} c_i \exp\left[-\sum_{j=1}^{6} a_{ij}(x_j - p_{ij})\right]$	6	$[0,1]^n$	-3.32
$f_{21} = -\sum_{i=1}^{5} \left(\sum_{j=1}^{4}(x_j - a_{ij})^2 + c_i\right)^{-1}$	4	$[0,10]^n$	-10.15
$f_{22} = -\sum_{i=1}^{7} \left(\sum_{j=1}^{4}(x_j - a_{ij})^2 + c_i\right)^{-1}$	4	$[0,10]^n$	-10.34
$f_{23} = -\sum_{i=1}^{10} \left(\sum_{j=1}^{4}(x_j - a_{ij})^2 + c_i\right)^{-1}$	4	$[0,10]^n$	-10.54

Table 3.6: Comparison of mean best fitness between LMSEP and MSEP, LEP, FEP, CEP

	Generations	LMSEP	MSEP	LEP	FEP [1]	CEP [1]
		mean best	mean best	mean best	mean best	mean best
f_1	1500	3.803e-5	6.209e-5	3.048e-2	5.72e-4	1.91e-4
f_2	2000	1.556e-3	8.226e-2	1.232e-2	7.60e-2	2.29e-2
f_4	5000	0.767	0.629	1.053e-3	0.3	2.0
f_6	1500	0	43.8	0	0	577.76
f_7	3000	3.514e-2	3.56e-2	9.197e-3	7.6e-3	1.8e-2
f_9	5000	61.39	63.44	42.064	4.6e-2	89.0
f_{10}	1500	1.956e-3	6.498	1.797e-1	1.76e-2	8.79
f_{11}	2000	5.0e-2	7.768e-2	5.5e-2	2.49e-2	8.13e-2
f_{12}	1500	1.282	9.36e-1	1.3e-2	9.2e-6	1.76
f_{15}	4000	2.247e-4	3.077e-4	4.5e-4	1.8e-2	9.2
f_{16}	100	-1.0316	-1.0316	-1.029	-1.03	-1.03
f_{18}	100	3	3	3.19	3	3
f_{19}	100	-3.863	-3.863	-3.863	-3.86	-3.86
f_{21}	100	-8.744	-8.62	-7.84	-5.50	-6.43
f_{22}	100	-9.585	-9.37	-9.68	-5.73	-7.62
f_{23}	100	-9.483	-9.63	-9.70	-6.41	-8.86

Table 3.7: Comparison of standard deviation between LMESP and MSEP, LEP, FEP, CEP

	Generations	LMSEP	MSEP	LEP	FEP [1]	CEP [1]
		Std. dev	Std. dev	Std. dev	Std. dev	Std. dev
f_1	1500	7.971e-5	1.607e-4	6.043e-3	1.3e-4	5.9e-4
f_2	2000	9.37e-4	4.346e-1	3.136e-3	7.7e-4	1.7e-4
f_4	5000	1.09	1	2.059e-4	0.3	0.5
f_6	1500	0	126	0	0	1125.76
f_7	3000	1.854e-2	1.744e-2	2.486e-3	2.6e-3	6.4e-3
f_9	5000	13.18	13.8	15.127	4.6e-2	1.2e-2
f_{10}	1500	1.762e-3	2.485	2.813e-2	2.1e-3	2.8
f_{11}	2000	4.843e-2	11.198	1.114e-2	1.8e-2	9.2
f_{12}	1500	2.065	2.46e-2	5.7e-5	3.6e-6	2.4
f_{15}	4000	1.885e-4	1.090e-6	1.5e-4	1.8e-2	9.2
f_{16}	100	4.817e-9	3.417e-8	3.247e-3	4.9e-7	4.9e-7
f_{18}	100	4.431e-8	0	2.183e-1	0.11	0
f_{19}	100	7.22e-8	4.568e-7	1.19e-4	1.4e-5	1.4e-2
f_{21}	100	2.744	2.59	2.934	1.59	2.67
f_{22}	100	2.136	2.32	1.959	2.12	2.95
f_{23}	100	2.466	2.38	2.284	3.14	2.92

according to a dynamic probabilistic distribution. CEP is an EP using a Gaussian mutation and FEP stands for EP using a Cauchy mutation which then is extended to LEP.

Table 3.6 lists the results of the mean best fitness generated in benchmark functions, from which it is observed that LMSEP produces a reasonable result at a similar level of precision which reveals that the mixed strategy performs at least as well as a pure strategy. However, it can be seen that, with the use of LEP or FEP, a considerably improved result for f_4, f_7, f_9 and f_{12} can be obtained. By exploring the mutation mechanism of LEP and FEP, the main reason can be considered as the relative higher flexibility of LEP and FEP than that of LMSEP. As the feature and mixed strategy is one-to-one correspondence in LMSEP, the update of mixed strategy would be stopped in certain generation once there is no improvement to the feature of local fitness landscape. As a result, the ability of producing large step jump is reduced at the later stage of the search [1]. It would be potentially favourable to assign a parameter, expressed as percentage, to let the process have a limit flexibility which is not in compliance with the one-to-one correspondence.

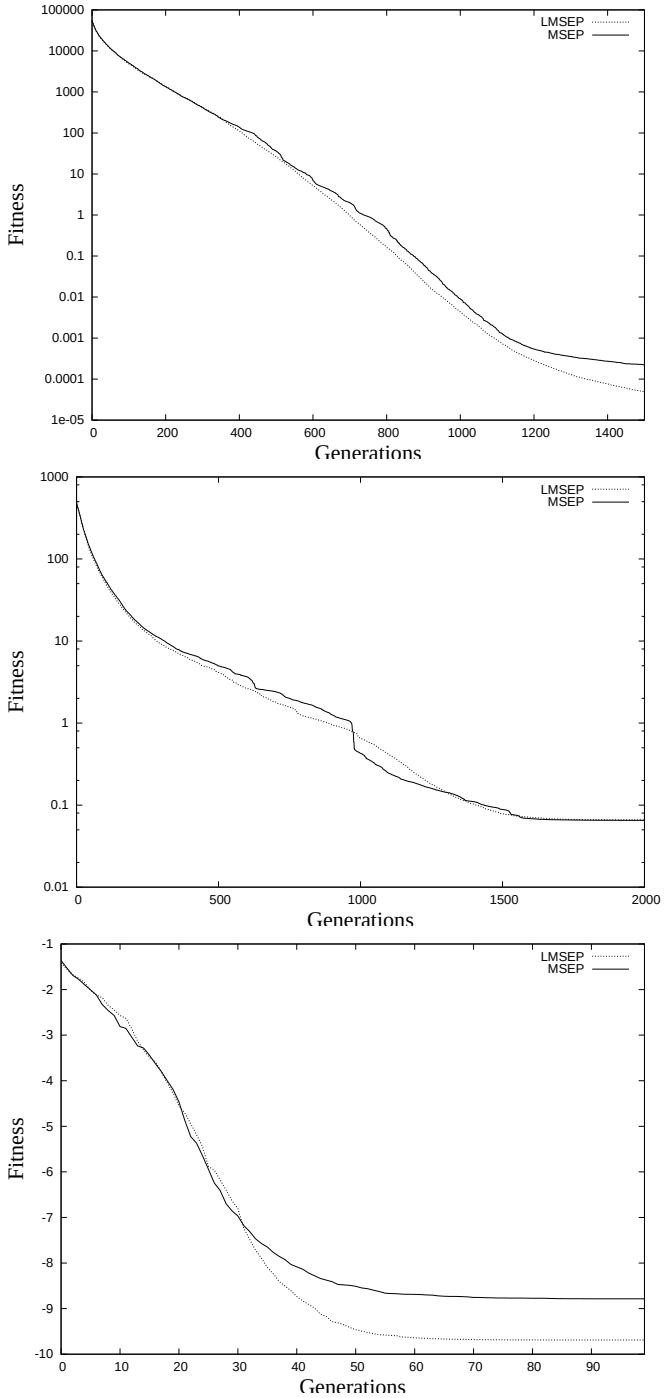

Figure 3.10: Comparison between LMSEP and MSEP.

Fig. 3.10 provides a graphical comparison of average values of population found by two mixed strategies, LMSEP and MSEP, over 50 runs, where both algorithms employ two types of mutation operator, namely Gaussian mutation and Cauchy mutation. Two benchmark functions, f_1, f_{11} and f_{22}, are tested here. It is clear that LMSEP can search further on f_1 and f_{22} while MSEP can not find relative improved results. For f_{11}, LMSEP displays a faster convergence rate than MSEP. LMSEP quickly reaches approximately 1 in around 1000 generations. After that, MSEP exhibits a substantial descent and overtakes LMSEP. However finally, both algorithms reach approximatively same result around 1700 generations. Take all cases into account, the curves of process suggest a more efficient progress is introduced by LMSEP.

The standard deviation of LMSEP whose evaluation is given in Table 3.7 is also compared with those obtained using other approaches. According to the results, LMSEP exhibits similar performances on most functions except f_4, f_7, f_9 and f_{12}. This fact indicates that LMSEP has at least the same stability with a single pure strategy. However, LMSEP does not present a better performance on some function to which LEP and FEP are nicely applied. It suggests that the search of mixed strategy is sometimes not able to focus on the position where the global minimum is situated. It means the adjustment of the feature of local fitness landscape, the implementation of φ in this work, remains to be carefully modified in future research so that a better result can be expected.

3.5 Summary

This chapter has presented a new EP, LMSEP, which is characterized by the local fitness landscape using a mixture of different mutation strategies. The approach addresses the drawbacks of conventional EPs that employ a single mutation operator. In addition, a training procedure has been given to promote LMSEP in an efficient and intelligent way, by introducing a self-learning algorithm.

The performance of LMSEP is firstly trained on 7 functions and then tested on a suite of 16 benchmark functions, in comparison with previously studied EPs. The experimental evaluation indicates that the new approach has the ability to produce relatively more acceptable results. The tests regarding the standard deviation also demonstrate that LMSEP has a more robust performance.

Although the training procedure leads to a fast performance, it may occasionally miss certain regions that should be checked for. To address this issue, a fine adjustment of training procedure remains an active research. For instance, a backward jump procedure may be potentially employed. As a compatible satisfactory result can be obtained using FEP and LEP, a better implementation of the φ parameter may be useful. Additionally, more mutation operators can be taken into account, (e.g. Lévy mutation). Finally, introducing mixed strategies to other types of operator like crossover and selection also forms an interesting piece of future work.

Chapter 4

Mixed Strategy based on Game Theory with Incomplete Information

In the previous chapter, the newly developed algorithm is primarily constructed with an effective analysis into the local fitness landscape. In addition, there are some different ways to design an algorithm which combines different mutation operators. One of them which is presented in this chapter is a modified mixed strategy (IMEP) involving a process with incomplete information. It will provide a possible comparison to the one using local fitness landscape.

4.1 Mixed Strategy Using Game Theory with Incomplete Information

Progress has recently been made in an effort to adjust mutation strategies [6, 7, 116, 119]. In previous studies [7, 9], the design of a mixed strategy mainly utilizes the payoff of each operator (i.e. an operator which produces a higher fitness will receive a better payoff). The payoffs to each individual and the history of the running process are fully known to each individual. This chapter, in contrast, proposes a novel mixed strategy that the payoff to each mutation is uncertain or not precisely determined beforehand.

A mixed strategy based on payoff is proposed in this chapter. Remind of the probability distribution $(\rho(s_1), \cdots, \rho(s_L))$ which maximizes the drift towards the global optima, i.e.,

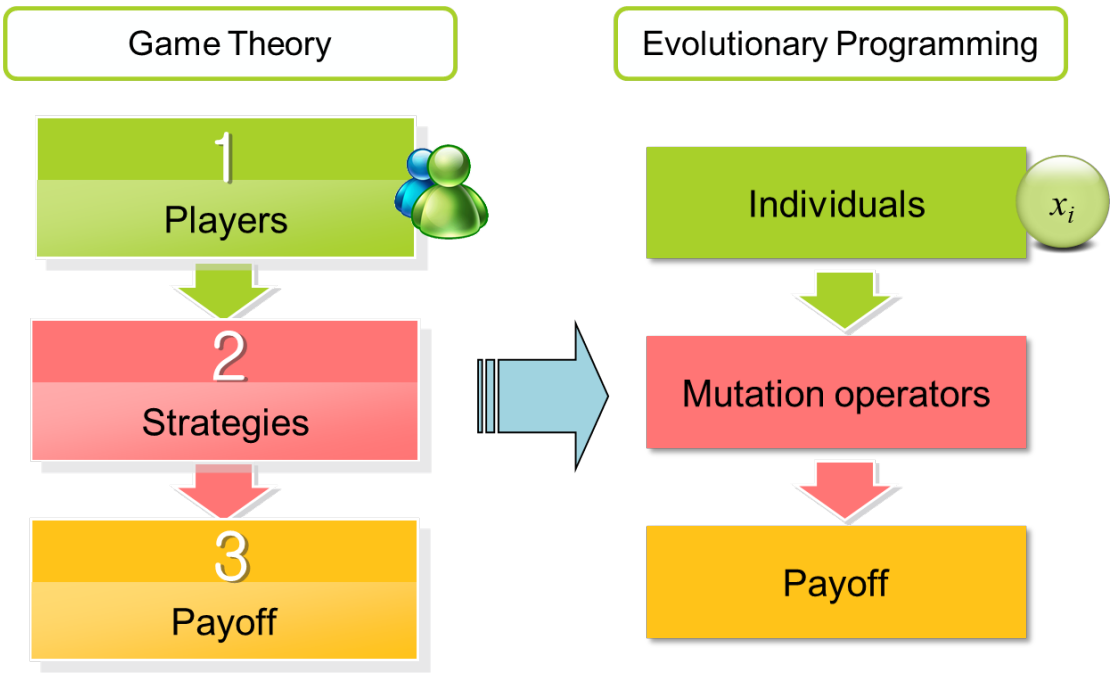

Figure 4.1: Transition from Game Theory to Evolutionary Programming

$$\max_{\pi}\{d(\vec{x}^{(t)},\vec{y});\ \vec{y}\in S_{\min})\}.$$

At every generation, a real number u_i is assigned to individual i in terms of the result of fitness evaluation. It is called the associated payoff to individual i. The probability distribution of strategy π is updated according to the combined payoff function u which is assigned to each mutation operator s and denoted by a vector $u(s)$. Previous approaches [6],[7], in general, mainly discuss a process of complete information, where the vector $u(s) = (u_1, \cdots, u_L)$ and the payoff function are known to all mutation operators at every generation. That is, the payoff function is explicitly determined by the performances of different mutation operators.

Given a simultaneous-move process of incomplete information, where only part of the payoff function assigned to each mutation operator is known, thus it is impossible to construct the required full payoff functions. Therefore, the payoff function assigned to strategy i is not denoted by a simple vector but by $u_i = (s_1, \cdots, s_L;\ p_i)$, where p_i is named strategy i's type and belongs to a set of possible types P_i. Each type p_i corresponds to a different payoff function that strategy i might have. This idea represents the

possibility that each strategy can be updated in more than one direction corresponding to its different types of payoff functions.

In light of this, a dynamic modification of updating the probability distribution of the mixed strategy can be generated. Suppose that in the process with incomplete information, strategy i's payoff function is no longer entirely known to others. In addition to existing well known factors, e.g. the jump distance produced by each mutation operator, crossover points and mutation probability, there may be a certain interfering element whose detailed information is unknown, such that function u_i is replaced by $F(u_i)$, where $F(u_i)$ is a small but implicitly modification of of u_i. Thus, the major task is to design a reasonable representation of $F(u_i)$.

Inspired by this observation, an Incomplete Information Factor (IIF) p is introduced to represent the unknown factor. Given such a p the deviation between $F(u_i)$ and u_i should not be too large such that they overwhelm the entire algorithm. The construction of p and in turn the construction of $F(u_i)$ in this chapter is based on a uniform distribution on $[0, \gamma]$. A simple procedure to adjust the mixed strategy is given as follows.

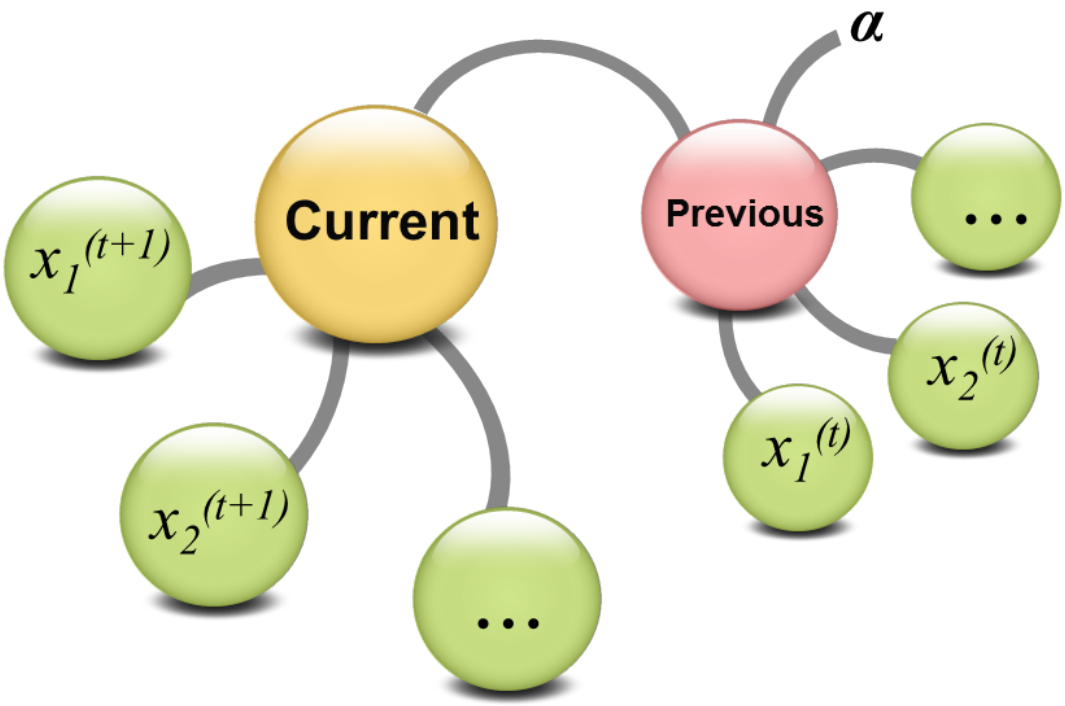

Figure 4.2: Consider the impact of history strategy

For a population $(x_1, \cdots, x_\mu)$,

1. Calculate the maximum jump distance an individual makes once a mutation process is applied successfully to it. Denote the parent individual by $x_i^{(t)}$ and the offspring by $x_i^{(t+1)}$.

$$d(x)_i^{(t+1)} = \begin{cases} \max\limits_{1 \le j \le n} \mid x_{i_j}^{(t+1)} - x_{i_j}^{(t)} \mid, \\ 0, \end{cases}$$

where the first function is taken if $f(x^{(t+1)}) < f(x^{(t)})$; otherwise, zero is taken. Based on the resulting distance value, the maximum jump distance induced by strategy s_1 is generated as follows:

$$d(s_1)^{(t+1)} = \max_i x_i^{(t+1)},$$

where $x_i^{(t+1)}$ is an individual whose strategy is s_1.

2. Consider the impact of history strategy which determine how much the previous movement will kept in memory to affect the payoff function.

$$d(s_1)^{(t+1)} = \begin{cases} d(s_1)^{(t+1)}, & \text{if } d(s_1)^{(t+1)} \ge \alpha \cdot d(s_1)^{(t)}, \\ \alpha \cdot d(s_1)^{(t)}, & \text{otherwise.} \end{cases}$$

3. Define the original payoff function u_i. Since there are only two mutation operators employed in this chapter, namely Gaussian mutation s_1 and Cauchy mutation s_2. u_1 and u_2 are determined by

$$\begin{cases} u_1(s_1, s_2) = \dfrac{d(s_1)}{d(s_2)}, \\ u_2(s_1, s_2) = \dfrac{d(s_2)}{d(s_1)}. \end{cases}$$

Introduce a controlling parameter β to avoid the case that u_1 or u_2 reaches zero, such that

$$u_1(s_1, s_2) = \begin{cases} \beta, & \text{if } \dfrac{d(s_1)}{d(s_2)} \le \beta, \\ \dfrac{1}{\beta}, & \text{if } \dfrac{d(s_1)}{d(s_2)} \ge 1/\beta, \\ \dfrac{d(s_1)}{d(s_2)}, & \text{otherwise.} \end{cases}$$

4. Construct the revised version of the payoff functions by introducing a uniform distribution. Currently, two methods of designing and applying the IIF have been taken into account: (1) substitute u_i with $p_a \cdot u_i$; (2) set u_i to be $u_i + p_b$, where p_a and p_b are independent draws from a uniform distribution on $[0, \gamma]$. As in a process involving incomplete information, p_a and p_b together form a possible type set P_i for strategy i.

5. For a mixed strategy consisting only of two mutation operators $S = \{s_1, s_2\}$, the algorithm for updating the probability distribution $\pi(\rho(s_1), \cdots, \rho(s_L))$ can now be established, employing a proportion of its own payoff among the total payoff as follows:

$$\rho_1(s_1) = \frac{F(u_1)}{F(u_1) + F(u_2)}$$

$$= \begin{cases} \dfrac{p_a \cdot u_1(s_1, s_2)}{p_a \cdot u_1(s_1, s_2) + p_a \cdot u_2(s_1, s_2)} \\ \dfrac{u_1(s_1, s_2) + p_b}{[u_1(s_1, s_2) + p_b] + [u_2(s_1, s_2) + p_b]} \end{cases}$$

where every p_a and p_b are randomly generated within $[0, \gamma]$. It means that the three p_a in the function are unlikely to take the same value.

The details of the above mixed strategy evolutionary programming is given as follows:

1. **Initialization:** An initial population is generated consisting of μ individuals at random, each of which is represented by two real vectors $\vec{x}_i^{(0)}$ and $\vec{\sigma}_i^{(0)}$ ($i \in 1, 2, \cdots, \mu$). Each $\vec{x}_i^{(0)}$ is a random point in the search space, and each $\vec{\sigma}_i^{(0)}$ is a vector of the coordinate deviations. Both vectors have n real-valued components: for $i = 1, \cdots, \mu$.

$$\vec{x}_i^{(0)} = (x_i^{(0)}(1), x_i^{(0)}(2), \cdots, x_i^{(0)}(n))$$

$$\vec{\sigma}_i^{(0)} = (\sigma_i^{(0)}(1), \sigma_i^{(0)}(2), \cdots, \sigma_i^{(0)}(n))$$

For all individuals, their mixed strategy is taken to be the same one, i.e. $\pi = (\rho^{(0)}(1), \rho^{(0)}(2),)$, where $\rho^{(0)}(1), \rho^{(0)}(2)$ represent the probabilities of choosing Gaussian and Cauchy mutation operators respectively. In the experiment, these are set to the same value initially i.e. 0.5.

2. **Mutation:** Denote t to be the generation counter. Each individual i chooses a mutation operator from its strategy set according to its mixed strategy $(\rho^{(t)}(1), \rho^{(t)}(2))$, and uses this strategy to generate a new offspring.

   The operator set includes the following two mutation operators. In each description individual parent i is written in the form $(\vec{x}_i^{(t)}, \vec{\sigma}_i^{(t)})$. The corresponding offspring i' is written in the form $(\vec{x}_{i'}^{(t)}, \vec{\sigma}_{i'}^{(t)})$.

   Gaussian mutation: Each parent i produces an offspring i' as follows: for $j = 1, 2, \cdots, n$

   $$\sigma_{i'}^{(t)}(j) = \sigma_i^{(t)}(j) \exp\{\tau_a N(0,1) + \tau_b N_j(0,1)\}$$

   $$x_{i'}^{(t)}(j) = x_i^{(t)}(j) + \sigma_{i'}^{(t)}(j) N_j(0,1)$$

   where $N(0,1)$ stands for a standard Gaussian random variable (fixed for a given i), and $N_j(0,1)$ stands for an independent Gaussian random variable generated for each component j. The control parameter values τ_a and τ_b are chosen as follows:

   $$\tau_a = 1/\sqrt{2\mu} \text{ and } \tau_b = 1/\sqrt{2\sqrt{\mu}}.$$

   Cauchy Mutation: Each parent i generates an offspring i' as follows: for $j = 1, 2, \cdots, n$

   $$\sigma_{i'}^{(t)}(j) = \sigma_i^{(t)}(j) \exp\{\tau_a N(0,1) + \tau_b N_j(0,1)\},$$

   $$x_{i'}^{(t)}(j) = x_i^{(t)}(j) + \sigma_{i'}^{(t)}(j) \delta_j$$

   where δ_j is a standard Cauchy random variable, which is generated anew for each component j. The parameters τ_a and τ_b are set to the values used in the Gaussian mutation.

   After mutation, a total of μ new individuals are generated. The offspring population is denoted by $I'(t)$.

3. **Fitness Evaluation:** Calculate the fitness of individuals in both parent and offspring populations.

4. **q-Tournament Selection:** For every individual $i \in 1, 2, \cdots, 2\mu$ in the parent and offspring populations, a winning function w_i is initialized to zero. For each individual i, select another individual j at random and compare fitness $f(i)$ with

$f(j)$. If $f_i < f_j$, then the winning function for individual i is increased by one ($w_i = w_i + 1$). This procedure is performed q times for each individual. Based on the winning function, μ individuals from parent and offspring population with highest winning values are selected in the next generation, denoted by $I(t+1)$.

5. **Adjustment of Mixed Strategy:** For each individual i in population $I(t+1)$, its mixed strategy should be adjusted as follows: Given a population $(x_1, \cdots, x_\mu)$, first, identify the largest jump distance produced by a mutation operator (Gaussian mutation and Cauchy mutation in this chapter). Then, apply the history parameter α and calculate the original payoff u_i. Afterwards, apply the IIF p to u_i to generate the final payoff. Based on it, the probability of mutation strategy i of next generation is established.

6. Steps 2-5 are repeated until given stopping criterion is satisfied.

4.2 Experimental Results and Analysis

The above mixed strategy EP (IMEP) is evaluated on 7 test functions, which were used to test IFEP in [1]. The description of these functions is given in Table 4.1. Among them, functions f_1 and f_2 are unimodal functions, and f_3 and f_4 are multimodal functions with many local minima, $f_5 - f_7$ multimodal functions with only a few local minima.

Parameters α in Eq. (3) and β in Eq. (3) are set to 0.9 and 0.05 respectively. Other parameter values in the mixed EP are taken to be the same as those in [1]. Population size $\mu = 100$, tournament size $q = 10$, and initial standard deviation is set to $\sigma = 3.0$. The stopping criterion is: to stop running at 1500 generations for functions f_1 and f_3, 2000 generations for f_2 and f_4, 100 generations for $f_5 - f_7$. The lower-bound used in this chapter is $\sigma_{\min} = 10^{-5}$ for all functions except f_4. Since f_4 has a larger definition domain than the rest, $\sigma_{\min}$ is taken to be a bigger value 10^{-4}. At the initial step, the mixed strategy distribution is set to (0.5, 0.5). Results for $f_1 - f_4$ are averaged over 50 independent runs, and for $f_5 - f_7$ over 1000 independent trials.

The performance of IMEP is compared with that of MEP, MSEP, LEP, FEP and CEP, whose results are presented in Table 4.2 and Table 4.3. MEP [6] is the original algorithm of IMEP, which is inspired from the game with complete information in game

	Test functions	domain	$f_{\min}$
$f_1 =$	$\sum_{i=1}^{30} x_i^2$	$[-100,100]^{30}$	0
$f_2 =$	$\sum_{i=1}^{30} \mid x_i \mid + \prod_{i=1}^{30} \mid x_i \mid$	$[-100,100]^{30}$	0
$f_3 =$	$-20\exp\left(-0.2\sqrt{\frac{1}{30}\sum_{i=1}^{30} x_i^2}\right)$	$[-32,32]^{30}$	0
	$-\exp\left(\frac{1}{30}\sum_{i=1}^{30}\cos(2\pi x_i)\right) + 20 + e$		
$f_4 =$	$\frac{1}{4000}\sum_{i=1}^{30} x_i^2 - \prod_{i=1}^{30}\cos(x_i/\sqrt{i}) + 1$	$[-600,600]^{30}$	0
$f_5 =$	$-\sum_{i=1}^{5}\left(\sum_{j=1}^{4}(x_j - a_{ij})^2 + c_i\right)^{-1}$	$[0,10]^4$	-10.15
$f_6 =$	$-\sum_{i=1}^{7}\left(\sum_{j=1}^{4}(x_j - a_{ij})^2 + c_i\right)^{-1}$	$[0,10]^4$	-10.34
$f_7 =$	$-\sum_{i=1}^{10}\left(\sum_{j=1}^{4}(x_j - a_{ij})^2 + c_i\right)^{-1}$	$[0,10]^4$	-10.54

Table 4.1: Seven test functions, where the coefficients of $f5 - f7$ are given in [1].

theory. MSEP [7] is an EP with a mixed mutation strategy in which four different mutation operators are considered and adaptively employed according to a dynamic probabilistic distribution. CEP and FEP are two pure strategy EP using a Gaussian mutation and a Cauchy mutation respectively.

Table 4.2 lists the results of the mean best fitness generated in benchmark functions, from which it is obvious that IMEP performs much better than three pure strategies, LEP, FEP and CEP, over all test functions except f_3. However, IMEP still produces a reasonable result at a similar level of precision which reveals that the mixed strategy performs at least as well as a pure strategy. The reason is that a mixed strategy algorithm has much higher flexibility in searching the solution space. However, when comparing with other mixed strategy algorithms, MSEP, IMEP does not present a better result. The main reason for this is that there are four types of mutation operation applied in the mixed strategy, so that the process of the experiment has a relative higher flexibility than IMEP presented in this chapter. Note that the lower bound $\sigma_{\min}$ used in the experiment is also an important factor with respect to these results.

The standard deviation of IMEP whose evaluation is given in Table 4.3, is also compared with those obtained using other approaches. According to the results, IMEP

		IMEP(a)	IMEP(b)	MEP [6]	MSEP [7]	LEP	FEP [7]	CEP [7]
	generations	mean best	mean best	mean best	mean best	mean best	mean best	mean best
f_1	1500	3.887e-5	1.487e-5	9.151e-6	1.0e-4	3.048e-2	2.3e-3	5.2e-4
f_2	2000	2.030e-3	2.681e-3	1.269e-3	4.1e-4	1.232e-2	8.1e-3	2.6e-3
f_3	1500	2.950e-4	3.591e-4	6.590e-4	1.7e-3	1.797e-1	5.2e-2	15.1
f_4	2000	2.534e-2	3.106e-2	1.706e-2	8.5e-4	5.5e-2	3.9e-2	8.6e-2
f_5	100	-8.558	-8.711	-8.774	-10.15	-7.84	-4.81	-5.54
f_6	100	-9.764	-9.681	-9.735	-10.4	-9.68	-5.91	-8.84
f_7	100	-9.748	-9.725	-9.841	-10.54	-9.70	-8.73	-9.58

Table 4.2: Comparison of mean best fitness between IMEP and MEP, MSEP, LEP, FEP, CEP

		IMEP(a)	IMEP(b)	MSEP [7]	LEP	FEP [7]	CEP [7]
	generations	Std. dev.	Std. dev.	Std. dev.	Std. dev.	Std. dev.	Std. dev.
f_1	1500	6.174e-5	2.016e-5	1.3e-5	6.043e-3	2.2e-3	5.4e-4
f_2	2000	4.267e-4	5.101e-4	2.1e-5	3.136e-3	7.7e-4	1.7e-4
f_3	1500	6.449e-5	6.577e-5	4.3e-4	2.813e-2	2.5e-2	2.6
f_4	2000	2.296e-2	2.757e-2	1.3e-3	1.114e-2	2.3e-2	0.12
f_5	1000	2.705	2.674	5.0e-5	2.934	0.18	1.48
f_6	1000	1.914	2.023	4.7e-6	1.959	1.57	1.41
f_7	1000	2.176	2.245	1.3e-4	2.284	0.87	0.68

Table 4.3: Comparison of standard deviation between IMEP and MEP, MSEP, LEP, FEP, CEP

		$\gamma = 1$	$\gamma = 2$	$\gamma = 5$	$\gamma = 10$
	generations	mean best	mean best	mean best	mean best
f_1	1500	1.487e-5	3.436e-5	1.596e-5	2.563e-5
f_2	2000	2.681e-3	2.871e-3	3.014e-3	5.254e-3
f_3	1500	3.591e-4	3.477e-4	3.973e-4	2.858e-4
f_4	2000	3.106e-2	2.695e-2	2.619e-2	1.828e-2
f_5	1000	-8.711	-8.737	-9.831	-8.834
f_6	1000	-9.681	-9.618	-9.596	-9.641
f_7	1000	-9.725	-9.730	-9.697	-9.688

Table 4.4: Comparison of mean best fitness of IMEP(b) with respect to different γ

exhibits a better performance on $f_1 - f_4$ and a similar performance on $f_5 - f_7$ in comparison to a pure strategy (LEP, FEP or CEP). This indicates that IMEP has a more stable performance on both unimodal and multimodal problems. However, IMEP does not present a better performance than MSEP. Having been illustrated previously, it is affected by the types of mutation strategy introduced in the experiment.

As can be seen from Table 4.2 and Table 4.3, the result of IMEP is very similar to that of its corresponding algorithm MEP evolving complete information. Another experiment has therefore been carried out. This attempt to trying to identify if it is affected by the parameter γ or a problem with the current design of IMEP. 4 different values of γ are applied to IMEP and the results of mean best fitness are presented in Table 4.4. Note that, except for the value of γ, all other parameters are set to the values described in the previous experiment.

As shown in the table, tests on the same function all produce results with the same level of precision, no matter what value for γ has been applied. Therefore, the algorithm is not sensitive to the change of IIFs. Furthermore, these results are similar to the ones generated by MEP on same testing functions. This suggests that our modification of the original algorithm, the introduction of IIF, did not exert a considerable influence on the overall performance. This is because the value of IIF introduced in the initial tests is kept within a minimum scope. The impact of the IIF will be increasingly enlarged if the value of IIF can be further configured, possibly with parameter control with a learning procedure.

4.3 Summary

This chapter has presented a new EP using mixed strategies, IMEP, to combat the drawbacks of conventional EPs that employ a single mutation operator. Efforts have been made in order to explain why and how LMSEP works, which is characterized by the local fitness landscape using a mixture of different mutation strategies.

The performance of IMEP is tested on a suite of 7 benchmark functions and compared with previously studied EPs. The experimental results confirmed that the new approach has the ability to perform at least as well as the best of different conventional strategies with single mutations. Furthermore, the tests regarding standard deviation also demonstrate that IMEP has a more stable performance.

We aim at introducing incomplete information from game theory to evolutionary programming. Since the current work is very initial, the algorithm has not been optimised. Therefore, many aspects remain to be considered. The design of a larger modification to MEP is a potential research area. In addition, a successful design of IMEP can be treated as a framework for introducing incomplete information to the mixed strategy algorithms. The experimental evaluation, in our next step, will be extended to more complex functions. Furthermore, it is of interest to consider a comparison between IMEP and a multialgorithm genetically adaptive method for single objective optimization (AMALGAM-SO) [120], which is a multimethod algorithm blending several evolutionary algorithms together.

Chapter 5

Immune Algorithm with Mixed Strategy for Protein Folding

A variety of optimization algorithms have been employed to solve different types of optimization problems. It was particularly recognized that one of those heuristic algorithms is Clonal Selection Algorithm from the Artificial Immune System (AIS) [121], which shows an efficient performance when different hypermutation are employed together. It enables a great improvement on the performance of the AIS, especially in local search. That is, since each operator may only be efficient on certain fitness landscapes, it is desired to apply multiple mutation operators simultaneously in order to tackle different situations.

However, the mutation operators in the original clonal selection algorithm for numerical optimization are used in a sequential manner that prohibits the ability of taking advantages of more flexible mutation strategies. It takes advantages of two efficient mutation operators, hypermutation and hypermacromutation, as well as two crossover operators to construct a powerful optimization algorithm with simple and easy implementation. This chapter presents a mixed strategy, based on the local fitness landscape of different types of numerical functions. Experiments show that the proposed algorithm possesses adequate balance between exploration and exploitation such that it is less likely to fall into local optimal and has a faster and better convergence.

The protein folding prediction is one of the most challenging problems in computational biology, molecular biology, biochemistry, and physics, as there are countless

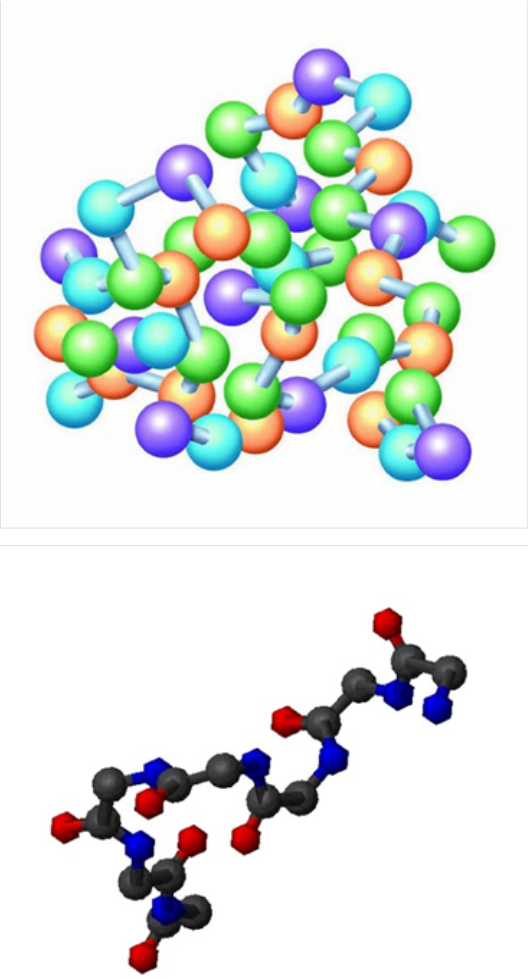

Figure 5.1: Protein molecule with sequence of amino acid

possible conformations for the backbone of a small protein [122]. Considering the complexity of protein folding problems, many researchers [122, 123] tend to focus on studying protein structure prediction in its simplified models or DillâĂŹs lattice models such as hydrophobic-polar (HP) model. It is actually based on the observation that the hydrophobic force is the main force that determines the unique native conformation and function state of small globular proteins.

In the standard HP model, each amino acid of protein is seen as a bead, and its connecting bonds as lines. In the HP model, proteins consist of a specific sequence of only two types of beads- H (hydrophobic/non-poplar) and P (hydrophilic/polar), which means the twenty amino acids can be categorized into two classes H and P. Also, the HP model takes into account the hydrophobic interaction as the main driving

force in protein folding, which has a unique native fold with an energy gap between the native and the first excited state Since environment inside cells are aqueous, these hydrophobic amino acids tend to be on the inside of a protein rather than on its surface; and meanwhile, the hydrophobicity is one of the key factors that determine how the chain of amino acids will fold up into an active or functional protein.

5.1 Modelling Protein Folding

5.1.1 Protein Structure

Proteins are, as shown in the first part of Figure 5.1, fundamental biological blocks which constructs every living organism. The functions of living beings are mostly carried out by functional biological proteins, where the proteins are bio-polymers assembled from a sequence of amino acid residues [124]. They are the main body, or backbone as shown in second part of Figure 5.1, of the protein's structure. The characteristics of a protein are specifically affected by the sequence since the sequence of amino acid is depended on the structure and the sequence. That is, the biological function of a protein is determined by the structure of the bio-polymers [125].

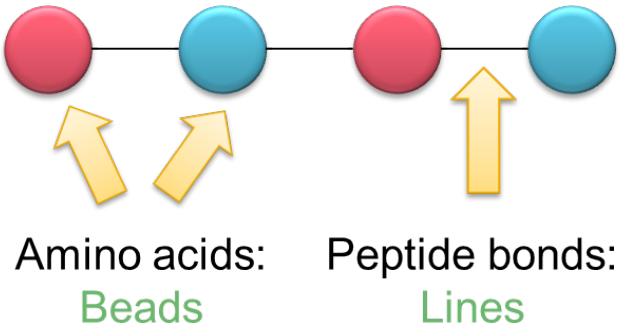

Figure 5.2: Amino acids

The structure of a protein, regardless of its function, is constructed by linking many amino acid monomer units via amide bonds (peptide bonds). These amino acid (residue) chains can vary in length, with chains of fewer than 50 residues often being called peptides, whereas bigger chains are referred to as proteins, where the term amino acid is shorthand for the common α-amino acid in biological living body.

5.1.2 Protein Folding Problem

One of the fundamental problem in computational molecular biology is the prediction of the protein folding and its resulting structure. The problem lies in biochemical physics and chemical biology, therefore it not only includes statistical mechanics, but also preserves the effects of evolution, which is one of the common features with most biological problems. Considerations must be made, with protein evolution in order to understand how mutational change in the amino acid sequence leads to structural and functional change [126].

The protein folding problem is the search for the most biologically active (functional) conformation of a protein (the native state), for a given sequence of amino acid residues. Before the pro, only the knowledge of its primary amino acid sequence is observed. That is, the one dimensional (1D) structure from which it is built. The problem is the prediction of the three dimensional (3D) local spatial arrangement (secondary structure) and the folded conformation (tertiary structure) adopted by a polypeptide molecule. It has been shown to be an NP-hard problem, in that no efficient algorithm can guarantee to find the native state [127]. If one is to understand how proteins fold and ultimately highlight the sequence-activity correlation of protein molecules, the relationship between sequence and structure is of critical importance and need to be analysed carefully.

5.1.3 Bioinformatics Techniques for Protein Folding

Protein folding is the physical process in which a linear polypeptide chain can be autonomously organized into a space-filling, compact, and well-defined three-dimensional structure [128]. The correct three-dimensional protein structure is essential to its biological function [129].

Although protein structure determination by biophysical techniques such as X-ray crystallography, cryoelectron microscopy and NMR has become highly automated and made considerable progress, the prediction of protein structure features is still one of major challenges in theoretical biophysics and bioinformatics [130]. The fundamental questions related to understanding protein folding mechanism are waiting for the proper answers, and the main problem concerning the vast potential complexity of cooperative interaction between amino acids remains to be resolved [130].

In order to improve the accuracy and efficiency of solving protein structure prediction, many scholars and researchers, [131, 129, 130] try every effort to study and develop some feasible algorithms or algorithmic strategies. Theoretical and computational protein folding studies in recent years have achieved some significant accomplishment in protein dynamics.

The reliability of natural proteins to fold to a unique, low energy, most stable state (native state) is related to the presence of a folding funnel on the free energy landscape, allowing mis-folded proteins to be guided towards the most energetically favourable conformation. To achieve a greater knowledge of protein folding dynamics, the nature of the free energy landscape must be understood. Although progress has been made over many decades, due to the complexity of the problem, it still remains unsolved [132].

5.1.4 Hydrophobic-Polar Model

There are a variety of protein models which differ in the way in which they approximate the protein molecule and how they treat the interactions between amino acid residues, and, if applicable, with solvents. Due to the enormous complexity and size of protein hyper-surfaces, models used to study the protein folding process tend to be simplified.

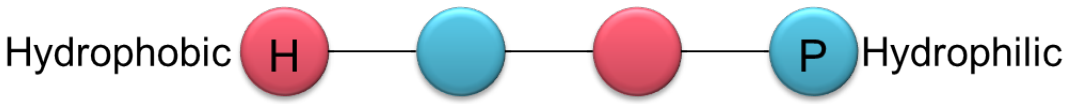

Figure 5.3: Two types of amino acids: hydrophobic and hydrophilic

The most simplistic of all models, the hydrophobic-polar lattice bead model (HP), has become one of the major tools for studying protein structure [133]. The basis of such model is that the hydrophobic force is primarily responsible for the determination of the unique native conformation and therefore the biological function of small globular proteins. Although simple, such models can still capture some essential features of he protein folding problem and provide a basis for thorough theoretical studies.

Moreover, the conformations in the HP model are restricted to self-avoiding walks on a lattice [122]. And the whole conformation is embedded in a two or three-D lattice, which simply divides spaces into amino acid-sized units and has bond angles keep

limited discrete values dictated by the structure of lattice like square, triangular or cubic. For instance, a 2-dimensional square lattice is typically used in the 2D HP model, while a 3-dimentional cubic lattice is generally applied to the 3D HP model.

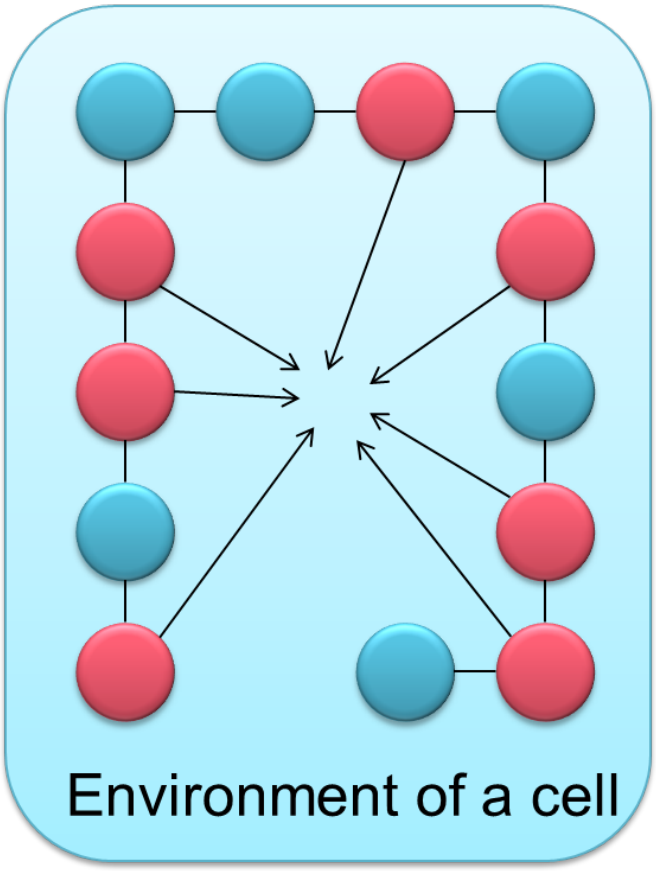

Figure 5.4: Driving force: hydrophobic interaction

- Hydrophobic amino acids (H beads) tend to come together to form a compact core that excludes water.
- hydrophilic amino acid (P beads) tend to face the outside.
- HP model involves attraction-interaction only

The twenty naturally occurring amino acids can be roughly classified into two categories based on their hydrophobicity. In the HP model, these two categories are exploited with amino acids categorised as either Hydrophobic (H) or Polar (P) residues. The primary amino acid structure of a protein, instead of comprising sequence of the

twenty amino acid alphabet, is therefore represented as a combination Hs and Ps, with each amino acid represented as a uniformly sized bead. The conformations of such a sequence are restricted to a self avoiding walk on a lattice, where lattice sites can only be occupied by a single bead. The presence of a lattice prevents bond lengths and angles from varying and thus both are constant throughout the use of this model.

The energy associated with any bead-bead interaction is described as a short range contact between topological neighbours. A topological neighbour is simply a pair of non-bonded beads that lie on adjacent lattice sites, i.e. they are not sequence neighbours. Interaction values for the possible topological contacts (local interactions) are:

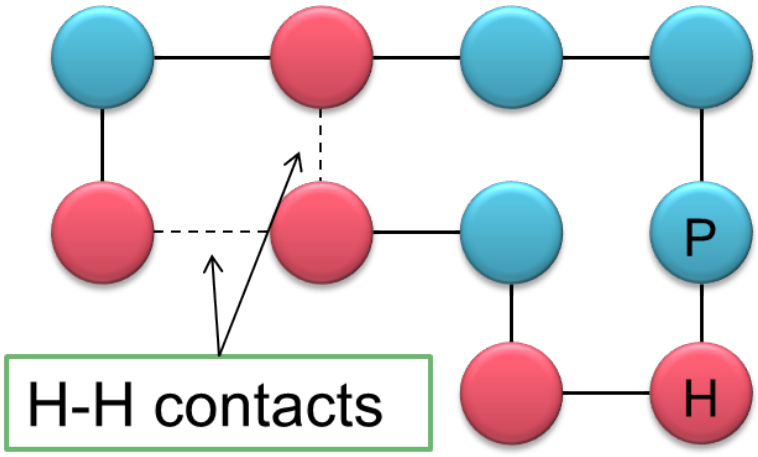

Figure 5.5: The H-H contacts of a protein sequence in the HP model

5.1.5 Free Protein Energy of the HP Model

The main method in HP model to evaluate the free energy of a conformation in the HP model is to count the interactions between beads, which are topological neighbours rather than sequential neighbours (non-local interactions). The simplest form of energy function counts the number of $H-H$ contacts. Each $H-H$ topological contact, that is, each lattice nearest neighbour $H-H$ contact interaction, has energy value $\varepsilon \leq 0$, while all other contact interaction types like $H-P$, $P-P$ contribute with $\varepsilon \geq 0$ to the total free energy. Consequently, any feasible conformation in the HP model can be assigned a free energy level and the residues interactions can be defined. The values of $H-H$, $H-P$ and $P-P$ interactions are:

$$\varepsilon_{HH} = -1, \tag{5.1}$$

$$\varepsilon_{HP} = \varepsilon_{PP} = 0. \tag{5.2}$$

So one can have the typical interaction energy matrix for the standard HP model accordingly. In the Dill's lattices model the native conformation is the one maximising the number of contacts $H-H$, i.e., the one that minimises the free energy function.

The total energy, **E**, of the conformation is found by summing the interactions [132]:

$$\mathbf{E} = \sum_{i \leq j} \varepsilon_i j \Delta_i j \tag{5.3}$$

where

$$\Delta_i j = \begin{cases} 1, & \text{if } i \text{ and } j \text{ are topological neighbours but not sequential neighbours,} \\ 0, & \text{Otherwise.} \end{cases} \tag{5.4}$$

The energy value for the conformation illustrated by Figure 5.5 is $E = -2$. In the process of founding the native conformation of the HP model, the number of $H-H$ interactions is maximised, therefore the free energy is minimised.

A solution, $r = \in \{F,L,R\}^{l-1}$, is represented by a sequence of relative directions (forward, left or right). It describes a self avoiding path through the lattice.

5.2 Evolutionary Algorithms for Protein Folding

Beutler and Dill [134] has researched algorithm specifically for the HP model of protein folding. It is called the Core-directed Growth (CG) method. Another approach is proposed by Toma, in which an algorithm specifically for the problem of protein folding

Table 5.1: Tortilla 2-D Benchmarks

Instance	Protein Sequence	l	E
1	$hphp_2h_2php_2hph_2p_2hph$	20	-9
2	$h_2p_2(hp_2)_6h_2$	24	-9
3	$p_2hp_2(h_2p_4)_3h_2$	25	-8
4	$p_3h_2p_2h_2p_5h_7p_2h_2p_4h_2p_2hp_2$	36	-14
5	$p_2h(p_2h_2)_2p_5h_10p_6(h_2p_2)_2hp_2h_5$	48	-23
6	$h_2(ph_3)ph_4p(hp_3)_2hp_4(hp_3)_2hph_4(ph)ph_2$	50	-21
7	$P_2h_3ph_8p_3h_10php_3h_12p_4h_6ph_2php$	60	-36
8	$h_12(ph)_2(p_2h_2)_2P_2h(P_2h_2)_2P_2h(P_2h_2)_2P_2(Ph)_2h_12$	64	-42
9	$h_3p_2(hp)_2hp_2(hp)_2hp_2h$	20	-10
10	$php_2hph_3ph_2ph$	85	-53
11	$hphph_3P_3h_4p_2h_2$	100	-48
12	$h_2p_5h_2p_3hp_3hp$	100	-50
13	$php_2hph_3ph_2ph_5$	18	-9
14	$hphph_3p_3h_4p_2h_2$	18	-8
15	$h_2p_5h_2p_3hp_3hp$	18	-4

called Contact Interactions (CI) method [135]. It is based on the standard Monte Carlo method and also there is something new about forming and preserving a hydrophobic core. This algorithm performed better than others. This is done by assigning a mobility to each residue in the protein. It has a low mobility and changes to this part if the conformation are less likely to be accepted.

An improved Pruned-Enriched-Rosenbluth Method (PERM) for the 2D HP lattice is presented by Hsu where the algorithm PERM covers many things except the Chain Growth (CG) method of Beutler and Dill [136].

Another usage of genetic algorithms is to find how energy conformations of HP model sequences. Unger and Moult's method with mutation operators was similar to a single MC step and a crossover operator [137]. The probability of accepting a solution is based on a cooling factor. Their research is put forward further by Konig and Danekar [138]. They use a simple genetic algorithm with a new crossover strategy and a niching technique. This so-called pioneer search is much faster and more reliable.

Krasnogor [129] and his colleagues proved that evolutionary algorithms optimization methods like genetic algorithms (GAs) are ideally reasonable algorithmic options to solve protein structure prediction problems by examining the design and application of genetic algorithms in dealing with the intractable protein structure prediction problems under the condition of the simple hydrophobic-hydrophilic model, which is also called HP model. They take into account of three basic algorithmic factors that may affect the effective application of genetic algorithms (GAs) in handling protein structure prediction problem, such as selecting and evaluating the commonly used representations and formulating the energy potential to secure the continuity of protein conformations.

To compare the ability of different optimization algorithms, a benchmark set of 15 protein sequences have been used to test over 20 algorithms [123]. Those algorithms encompass a wide range of evolutionary algorithms, from an improved ant colony optimization (ACO) algorithm, to some hybrid algorithms, including a genetic algorithm with tabu search and a memetic algorithm which self-adaptively selects from a storage of local search heuristics. It is called the *Tortilla 2-D HP Benchmarks*. To compare the performance of the new algorithm, part of it is being used. In Table 5.1, l is the length of the sequence and E^* is the optimal or best known energy value.

5.2.1 Hybrid Evolutionary Algorithms for Protein Folding

Liang and Wong apply a hybrid of evolutionary techniques and the Monte Carlo method in an single algorithm for protein folding, which is called Evolutionary Monte Carlo (EMC) [139]. EMC creates a population of candidate conformations where the probability of accepting a worse accepting a worse solution is dependent on a *temperature* and the temperature is lowered following each iteration. The new candidate solutions are created from the current population, using mutation and crossover operators based upon genetic algorithms. The EMC algorithm managed to perform better than the genetic algorithm of König and Danekar and another metropolis-based MC because of its lower energy conformations and faster speed.

A hybrid search algorithm combining genetic algorithm and tabu search is proposed by Jiang for the 2-dimensional HP model [140]. They argued that the proposed algorithm performed better than Monte Carlo, Evolutionary Monte Carlo and simple genetic algorithm.

Shmygelska and Hoos put forward an advanced Ant Colony Optimization (ACO) approach to the HP lattice model [122]. It is similar to the state-of-art evolutionary and Monte Carlo algorithms. The advance of this algorithm is long range moves to relax compact conformations and escape from local optima and improving ants that selectively take the best global solution found so far and apply a local search thus reducing computational cost.

5.3 Immune Algorithms for Protein Folding

An novel immune algorithm is proposed by Cutello for the protein structure prediction problem which also provided the Tortilla 2-D HP Benchmarks [123]. They use an aging operator and two specific mutation operators based on the clonal selection principal of the biological immune system. After testing on different circumstances such as the 2D and 3D HP square lattice model and the functional model protein, the immune algorithm was verified a competitive state-of-art algorithm.

5.3.1 Biological Immune System

Biological immune system acts as the major defensive line in terms of detection and reaction of abnormal intrusion into the organism of humans. There are a variety of different components either cooperatively or competitively contributing to the defensive mechanism [19], which are briefly summarised as follows:

What is known as the innate immune system behaves as the first line of defence when there exist intrusion caused by micro-organism. Two cells that forms the basic components of innate immune system are macrophages and neutrophils, which have structures called receptors that can be bind to specific molecular patterns commonly found in a range of different micro-organisms. Another function of these cells is that they also release molecules called cytokines which will signal the human body to activate inflammatory response. Certain functions of Cytokines and inflammatory are:

- Phagocytes are attracted by them to the cites of infection where a type of white blood cells called phagocytes can ingest and digest microbes or other intrusive materials into the human body.
- Foreign microbes are coated with protein in order for the phagocytes to digest them.
- Blood flow to the infection site is increased so that more immune cells will be transported to the infected area to deal with microbes or other foreign materials.
- Directly damage some foreign cells, bacteria and viruses.
- Increases in body temperature follows, as a result the activities of some pathogens slow down while the activities of immune cell are strengthened.

Adaptive immune system operates at a more advanced level than innate immune system, essentially formed by lymphatic system. Two types of lymphocyte are identified, B cells and T cells, which perform distinctive, but complementary roles in the lymphatic system. It is found that Lymphocytes only respond to infections when there is a inflammatory response from the innate immune system in presence [141].

A variety of cells are produced during clonal expansion. For example, cloned B cells can develop into 2 types, either become plasma cells that is able to produce antibodies

at a high rate, or memory cells, which circulate along lymphatic system through the entire human body and is ready to proliferate if re-infection is detected. On the other hand, T cells also differentiate themselves into different pathways [141].

5.3.2 Clonal Selection Algorithm

The clonal selection algorithm is an optimization algorithm. It is based upon the clonal selection principle in the biological immune system. This idea has been successfully applied to the protein structure prediction problem in [123]. In this application, the clonal selection algorithm models the sequence of amino acids as an antigen and the solution as B-cells.

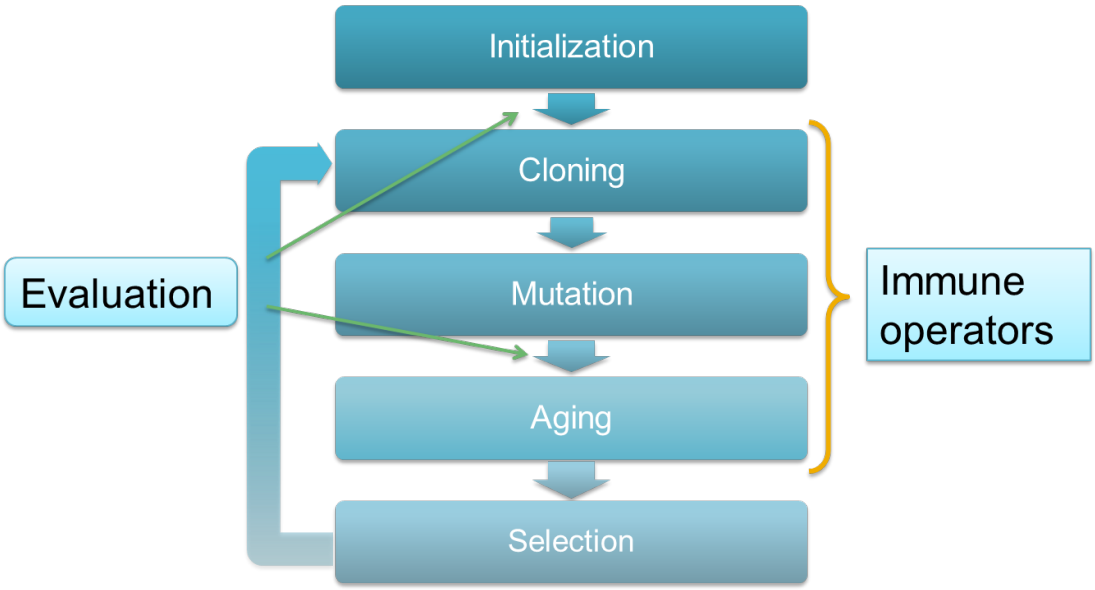

Figure 5.6: Conventional Clonal Selection Algorithm

As a key element in Clonal Selection Algorithm for a variety of optimization problems, mutation operators have attracted significant attention in research. The original mutation operator is typically Hypermutation, which usually lacks the ability of performing further search on some certain region. Therefore, different version have been developed, namely inversely proportional hypermutation, static hypermutation and hypermacromutation operators. In particular, hypermacromutation is proposed to extend the region of the parameter surface with high success rate values [123].

However, no single search algorithm is best on average for all problems [4], be they self-adaptation or not, which suggests the algorithm of the previous immune algorithm can be further improved if more effective mutation strategies could be added into the algorithm. However, as the mutation operators are utilized in a sequential way in the previous approach, it would be a really time-consuming task to follow this idea and prevent the algorithm from combining more mutation strategies.

According to [123], the two new search operators introduced in this project have both been designed to try and overcome these large energy barriers between minima. The mixed strategy uses a probability distribution to choose between hypermutation, hypermacromutation, one point crossover and uniform crossover. This allows for the possibility of large jumps in the search space via crossover and hypermacromutation, so that solutions stuck in local minima may have to opportunity to escape their current basin of attraction.

5.4 Mixed Strategy Applied to Clonal Selection Algorithm

The mixed strategy is inspired by different cell types in the immune system that work competitively and co-operatively to identify and remove foreign organisms from the host body. The mixed strategy can create an offspring from a clone by either mutations or crossover. Which strategy is chosen is decided by a probability distribution,

$$p = (p_{hyp}, p_{macro}, p_{1pt}, p_{uni}),$$

where p_{hyp} is the probability of creating offspring with the help of hypermutation, p_{macro} is the probability of creating an offspring by hypermacromutation, p_{1pt} is the probability of creating an offspring by one point crossover and p_{uni} is the probability of creating an offspring using uniform crossover. The probability distribution is updated according to the fitness of recently produced solutions. In the clonal selection algorithm the sequence of amino acids, $s \in \{h,p\}^l$, where l is the length of the sequence, models the antigen and the candidate solutions, $r \in \{F,L,R\}^{l-1}$, models the immune cells.

The operator to create the initial population, the cloning operator, aging operator and selection operator are the same as those used in [123]. After the initialisation step, the mixed strategy is applied to the genetic operators, including mutation and crossover.

The implementation of mixed strategy goes through each cell of the clone population, $P(clo)$, and assigns a mutation or crossover strategy according to the probability distribution p. If assigned the inversely proportional hypermutation operator the cloned cell is mutated in the way described in [123], whereby the number of mutations made to the parent solution is inversely proportional to its fitness. If assigned the hypermacromutation operator the number of mutations does not depend on any constant parameters or fitness value; the number of mutations, like the newly assigned directions, are randomly chosen.

If the cell is assigned the one point crossover operator it becomes the first parent. The second parent is chosen by *roulette wheel* selection, where the probability of choosing cell i, p_i, is the fitness of cell i, f_i, over the total fitness of the population:

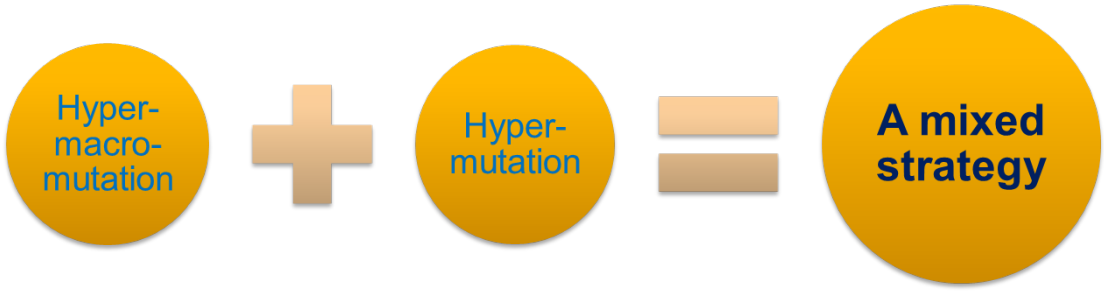

Figure 5.7: Clonal Selection Algorithm with Mixed Mutation Strategy

The crossover point is chosen with uniform probability. If the selected crossover point does not produce a feasible solution a new crossover point is chosen. If no point can result in a feasible solution then a new second parent is selected; this continues until a feasible solution is created and returned to the offspring population. It is possible that a clone of the current solution is selected to be the second parent, however this is not a problem because it stops the program from becoming stuck in an infinite loop if a feasible solution can not be produced using another member of the population and because duplicate solutions are deleted by the selection operator.

If the mixed strategy assigns uniform crossover to the cell then it becomes the first parent. The second parent is chosen with uniform probability from the clone population. From which parent a gene (direction) is taken from is chosen by a probability

proportional to its fitness; the probability of taking a gene, from parent one, p_{one}, is equal to the fitness of parent one, f_{one}, over the sum of the fitness of parent one and parent two:

$$p_{one} = \frac{f_{one}}{f_{one} + f_{two}} \tag{5.5}$$

As with the one point crossover operator, if uniform crossover does not produce a feasible offspring using parent two it will choose a new second parent from the population. Again it s possible for a solution to be crossed with itself but this stops the program from becoming stuck in a loop and duplicate solutions are deleted by the selection operator.

All strategies are initialised with equal probability such that the probability of mixed strategy are uniformly distributed:

$$p = (p_{hyp}, p_{macro}, p_{1pt}, p_{uni}),$$

where the number of the probabilities are initialised with

$$p = (0.25, 0.25, 0.25, 0.25).$$

From the 2nd generation onward, the strategy that produces the most offspring selected for the next generation is given a payoff. This means the probability of this particular strategy to be selected is increased whilst other strategies are assigned with decreased probabilities. The first use of this kind of algorithms was first on real-valued representation for optimisation problems [7].

In [7] each member of the population has its own probability distribution that chooses between Gaussian, Cauchy, Lévy and single-point mutation operators. If the offspring created using a particular mutation are selected for the next generation then the probability associated with that mutation is increased; if it is not selected than the strategy is penalised and the probability reduced. Offspring inherit their parents' probability distribution. This technique is inspired by the immune system where the cells with the highest affinity clone more rapidly. The technique has been adapted for this project so that all the solutions share the same probability distribution.

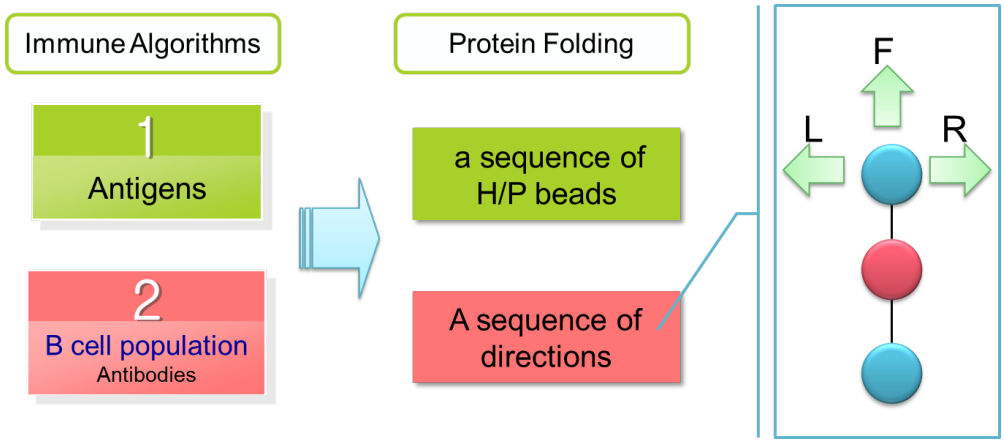

Figure 5.8: From Clonal Selection Algorithm to protein folding

When an offspring solution is created the operator used is recorded in its genotype (i.e. hypermutation, hypermacromuation, one point crossover or uniform crossover). Initial and new solutions are assigned an operator with uniform probability. After the selection of the best d solutions from the offspring population, the probability of the operator, h, that has created the most offspring, $p(h)$, is updated using:

$$p^{(t+1)}(h) = p^t(j) - p^t(j)_\gamma \tag{5.6}$$

The probabilities of the other operators, j, are updated using:

$$p^{(t+1)}(j) = p^t(j) - p^t(j)\gamma \tag{5.7}$$

The value of parameter $\gamma \in (0,1)$, which is used to control the probability distribution $\vec{p}$. It is chosen to between "0" and "1", which allows the normalisation condition to be meet.

$$\sum_{k=0}^{4} p(k) = 1 \tag{5.8}$$

where k includes the best strategy h and all other strategies j.

If one probability is greater than 0.65, meaning the associated operator is dominating the others, then the probability distribution is reassigned as: $\vec{p} = (0.25, 0.25, 0.25, 0.25)$. Re-setting the probabilities ensures that the probability of an operator being selected is dependent on its recent performance and not its past performance.

5.5 Experimental Results and Analysis

The clonal selection algorithm with the mixed strategy was tested on the Sequences 1-6, 9, and 13-15 in Table 5.1. The remaining five sequences could not be tested due to constraints on time and the amount of computational power available.

Each experiment was repeated 30 times. The success rate (SR) is the percentage of the 30 runs that find the lowest known energy value; the mean number of energy function evaluations (AES) is averaged over the successful runs. The maximum life span of a normal B cell (solution) $\tau_B = 1$ and the maximum life span of a B-memory cell $\tau_B = 5$. The value of the hypermutation parameter $c = 0.4$ for Sequences 1, 2, 3, 4 and 15 and $c = 0.5$ for Sequences 5, 6, 9, 13 and 14. The maximum number of function evaluations per run was $10,000,000$ and the maximum number of generations per run was $20,000$.

The first test of the clonal selection algorithm with mixed strategy, shown in Column B of Table 5.2, did not perform well with decreased success rate on Sequences 4 and 5, an increase in the number of energy function evaluations for Sequences 1 to 14 and was unable to find any results for Sequence 15. The population size was $d = 10$ and the cloning parameter was $dup = 4$. These values were based on the results from the mini-project, which in turn were based upon [123]. For the mini project the number of clones per solution was set to 2, one for each operator that created offspring. This was the basis for the choice of dup under this test; there are four potential operators that create offspring when using mixed strategy. However, because of the probabilistic way that operators are selected when using the mixed strategy, this size of clonal population was probably too small to allow all the operators to create offspring.

For Column C of Table 5.2 the cloning parameter dup was increased to 8 and the population size d remained 10. These results show a large improvement of 80% in the success rate of Sequence 5 compared to the clonal selection algorithm with crossover (Column A) and a moderate improvement in the success rate on Sequence 6 of 6.66%. These parameter settings could find solutions to Sequence 15 without becoming trapped. There is also a decrease in the number of function evaluations for Sequences 5, 6 and 13. The mixed strategy allows for large jumps between local minima; increasing the number of clones that the operator acts upon increases the number of chances for each operator to make these jumps, leading to improvements in success rate and number of energy function evaluations.

In the experiments shown in Columns D and E of Table 5.2 the population size and cloning parameter were increased further. The initial intention was that by increasing the clonal population the performance would improve further by creating even more chances for the operators to make jumps between minima. However, increasing the clonal population size resulted in increases in the number of energy function evaluations for Sequences 2, 9, 13, 14 and 15 and a drop to zero in the success rate for Sequences 3, 4, 5, 6. This drop in success rate is probably due to the increase in function evaluations exceeding the maximum energy evaluations allowed. It is possible that the clonal selection algorithm with mixed strategy would perform better with a large population size if experimental parameters, such as the maximum number of function evaluations, were adjusted.

Table 5.3 compares the clonal selection algorithm with mixed strategy to other published results. Comparison of Columns A and D show that Mixed Strategy decrease the average number of function evaluations on nine out of ten sequences. The new operator also increases the success rate by 43.33% for Sequence 5. Column B shows the results from [132]'s genetic algorithm; the clonal selection algorithm with mixed strategy decreases the number of function evaluations for five out of six sequences and improves the success rates for Sequences 4 and 5 by 30% and 87% respectively. Ant colony optimisation algorithm (Column C) is a very successful algorithm for the protein structure prediction problem on the HP model. [122] The results shows that the clonal selection algorithm with mixed strategy has matched its success rate on the six sequences, which proves mixed strategy can at least perform at the same level of the best known pure strategy.

5.6 Summary

This chapter focused on applying mixed strategy into the simple HP lattice model of protein folding problem. The native conformation of a protein is postulated to correspond to the shape that minimises the free energy. Because of this the protein structure prediction problem can be considered an optimisation problem. Many optimisation techniques have been applied to the problem including chain growth methods, [134, 135], variations of genetic algorithms [136, 138, 140, 142, 143, 144, 145], a memetic algorithm [129], ant colony optimization [122] and the clonal selection algorithm [123].

Many of the optimisation techniques applied to this problem have struggled with longer sequences in the tortilla benchmark set, especially those with tight hydrophobic cores (e.g. Sequence 5). In this project, mixed strategy is introduced into the algorithm, which is designed to overcome high energy barriers between local minima and increase population diversity in order to explore new parts of the search space. Both are inspired by processes in the biological immune system.

The mixed strategy is based upon the cell diversity in the immune system. Which crossover or mutation technique is used to create offspring depends on a probability distribution $p = (p_hyp, p_macro, p_1pt, p_uni)$. The technique that creates the most offspring selected for the next generation is rewarded with a payoff that increases its probability of being selected in the future. The mixed strategy helps overcome the large energy barriers between low energy conformations of the HP-model proteins by allowing for large jumps in the search space, or tunnelling, between minima.

Experimental tests show that the employment of mixed stratey improved the success rate on Sequences 4, 5, 6 and 15 when compared to the results from the mini-project and improved the success rate by 43% for Sequence 5 when compared to [123] and by 87% when compared with [143]. The success rate is the same as reported by [122].

The experimental results show that the clonal selection algorithm with mixed strategy is suitable for long sequences where the native conformation has a tight hydrophobic core. For shorter sequences, without large energy barriers between the local and global minima, the clonal selection algorithm with crossover is more suitable.

Table 5.2: Comparing algorithm with only crossover (Column A) and algorithm with mixed strategy (Columns B-E), for different values of d and dup, on success rate (SR) and average number of function evaluations used to find solution (AES)

	A		B		C		D		E	
	C, $d = 10$, $dup = 4$		M, $d = 10$, $dup = 4$		M, $d = 10$, $dup = 8$		M, $d = 20$, $dup = 20$		M, $d = 10$, $dup = 100$	
	SR	AES	SR	AES	SR	AES	SR	AES	SR	AES
Sequence 1	100.00	1841.52	100.00	42584.71	100.00	15218.35	100.00	5021.81	100.00	13299.05
Sequence 2	100.00	5110.30	100.00	128740.04	100.00	25319.60	100.00	17829.04	100.00	43003.28
Sequence 3	100.00	81374.97	100.00	526137.88	100.00	25918.25	83.33	877932.50	0.00	0.00
Sequence 4	93.33	259718.34	60.00	2580315.23	100.00	519930.49	0.00	0.00	0.00	0.00
Sequence 5	0.00	0.00	0.00	0.00	80.00	937472.41	0.00	0.00	0.00	0.00
Sequence 6	93.33	343798.54	26.67	3408430.35	100.00	90393.74	0.00	0.00	0.00	0.00
Sequence 9	100.00	3715.68	100.00	99.85.01	100.00	5139.65	100.00	4092.62	100.00	16849.58
Sequence 13	100.00	19159.29	100.00	241115.30	100.00	15883.18	100.00	34617.00	100.00	36552.17
Sequence 14	100.00	582.27	100.00	41998.19	100.00	16295.33	100.00	26623.09	100.00	18029.16
Sequence 15	-	-	-	-	100.00	248529.24	100.00	20718.66	100.00	48292.30

Table 5.3: Comparing Between Immune Algorithm with Mixed Strategy and other state-of-art optimisation algorithms

	A		B		C		D		E	
	Immune Algorithm		Genetic Algorithm		ACO		M, $d = 20$, $dup = 20$		M, $d = 10$, $dup = 100$	
	SR	AES	SR	AES	SR	AES	SR	AES	SR	AES
Sequence 1	100.00	23710	100.00	18338	100.00	-	100.00	15218.35	100.00	13299.05
Sequence 2	100.00	69816.7	100.00	27278	100.00	-	100.00	25319.60	100.00	43003.28
Sequence 3	100.00	269513.9	100.00	35128	100.00	-	83.33	25918.25	0.00	0.00
Sequence 4	100.00	2032504	70.00	113667	100.00	-	0.00	519930.49	0.00	0.00
Sequence 5	56.67	6403985.3	13.00	261311	100.00	-	0.00	937472.41	0.00	0.00
Sequence 6	100.00	778906.4	100.00	97691	100.00	-	0.00	90393.74	0.00	0.00

Chapter 6

Conclusion

During the last several decades, many kinds of population based Evolutionary Algorithms have been developed and considerable work has been devoted to computational methods which are inspired by biological evolution and natural selection, such as Evolutionary Programming (EP) [94], Artificial Immune System [19] and Differential Evolution (DE) [146]. The objective of these algorithms is not only to find suitable adjustments to the current population and hence the solution, but also to perform the process efficiently. However, a parameter setting that was optimal at the beginning of the algorithm may become unsuitable during the evolutionary process. Thus, it is preferable to automatically modify the control parameters during the runtime process. [29]. The approach required could have a bias on the distribution towards appropriate directions of the search space, thereby maintaining sufficient diversity among individuals in order to enable further ability of evolution.

6.1 Summary of Thesis

This thesis has offered an initial approach to developing this idea. The work starts from a clear understanding of the literature that is of direct relevance to the aforementioned motivations. The development of this approach has been built upon the basis of the fundamental and generic concepts of evolutionary algorithms. It shares with the common principles underlying the existing literature, especially regarding the representation of

population and individuals, as well as and the key evolutionary computational operations such as mutation, crossover and selection.

The work has exploited and benefited from a range of representative evolutionary computational mechanisms, including evolutionary programming, ant colony optimization, and clonal immune algorithm from artificial immune systems. In particular, essential issues in evolutionary algorithms such as parameter control, including the general aspects of parameter tuning and typical means for implementing parameter control have been investigated. This has given rise to the recognition of the technical significance of self-adaptive parameter control where the idea of combining different algorithms together is exhibited.

The idea of enabling an evolutionary algorithm to be self-guided, that is, to be capable of choosing the right method in a given situation has motivated the current research. In particular, both the hyper-heuristic algorithm and the memetic algorithm, which are established in the literature have set up a comparative work for the present development. Inspired by the appreciation of the potential benefits of utilizing mixed strategies in evolutionary algorithms, this work has developed several novel techniques that contribute towards the advancement of evolutionary computation and optimization.

One such novel approach is to construct a mixed strategy based on the concept of local fitness landscape. It exploits the concepts of fitness landscape and local fitness landscape. This strategy helps reinforce conventional evolutionary programming from two significant viewpoints. For this, the work has first applied local fitness landscapes to aid in the determination of the behaviour of mutations in evolutionary programming. Second, the work has introduced a training procedure that employs typical learning functions to determine the preferable probability distribution of mixed mutation operators, with respect to various types of local fitness landscape.

Both theoretical description and experimental investigation of this local fitness landscape based mixed strategy have been provided, and systematic comparisons with alternative approaches carried out. Supported with an analysis of the experimental results, the work has shown that the proposed approach can successfully overcome the major limitations of using conventional evolutionary programming methods that employ just single mutation operators.

Based on the aforementioned initial work, the thesis has presented a further improved approach to the development of a mixed strategy. This has been achieved by

exploiting game theory with the use of incomplete information. This new approach has been compared to the other through systematic experimental evaluation, on the same footing using the same set of test functions.

The experimental results have once again demonstrated that this further improved algorithm can successfully combat the shortcomings of conventional evolutionary programming methods that employ a single mutation operator. The new approach has proven to perform at least as well as the best of different conventional strategies with single mutations. The results have further shown that the proposed approach is able to possess a more stable performance while in action.

Another contribution of this thesis is the innovative application of mixed strategy. In particular, this work has extended the domain of usage of mixed strategy to from continuous to discrete problems. Given similar features to those associated with numerical function optimization, which is based on the different types of local fitness landscape, mixed strategy has been applied to artificial immune algorithms. This is facilitated by encompassing two crossover operators into the mixed strategy, one point crossover and conventional uniform crossover.

Such an improved method has been shown to be simple and easy for implementation. The work has been utilised to deal with the problem of protein folding in bioinformatics. Experimental results have been given, demonstrating that the proposed algorithm possesses an appropriate balance between exploration and exploitation. The use of this improved algorithm is less likely to fall into local optimal, entailing a faster and better convergence in resolving challenging realistic application problems.

6.2 Future Work

As indicated above, this thesis has achieved a number of important technical objectives in improving evolutionary computation methods, via the use of mixed strategies, including those proposed ones herein. However, the work carried out so far also gives rise to a number of important issues that may challenge the general success in developing more effective and efficient evolutionary algorithms. This final section of the thesis discusses several identified possible improvements over the present research, including ideas for both short-term and long-term developments.

6.2.1 Short-Term Improvements

There are many aspects that still to be addressed in the future. Researchers should extend the experiment to more types of problem and more complicated functions. Also a fine adjustment of local fitness landscape needs to be considered as an important part. MSEP leads to a compatible satisfactory so that a better implementation of the feature, λ, may be valuable and might lead to more stable performance. Additionally, more mutation operators can be taken into consideration, (e.g. Lévy mutation). Introducing mixed strategies to other types of operator is also worth to be considered.

For Mixed Strategy discussed in Chapter 4, incomplete information from game theory and evolutionary programming can be further extended in the future work. It is because the current work we are doing is still on a basic level; the algorithm has not been optimised. I can be aimed to consider more aspects such as the design of a larger modification to Modified Evolutionary Programming to perfect the work. In addition, a successful design of IMEP should be introduced as a framework for introducing incomplete information to the mixed strategy algorithms. In our next step, the experimental evaluation will be extended to more complex functions. A comparison between IMEP and a multialgorithm genetically adaptive method for single objective optimization (AMALGAM-SO) [120] can also be considered. It is a multimethod algorithm that combines several evolutionary algorithms together.

The experiments in the Protein Folding can also be designed further, making mixed strategy more adaptive to the application. To allow the algorithm sufficient opportunity to find the lowest known energy values, the maximum number of function evaluations should be increased. For those new operators, these longer sequences would be a favourable test because of the tight hydrophobic cores of the known native conformations. Further test of the clonal selection algorithm with mixed strategy and archive operators should be implemented on all benchmark sequences in order to find the best values for the algorithmâĂŹs parameters. Besides, the use of clonal selection algorithm with better tuning can help solving the protein structure prediction problem on models with even higher resolution.

6.2.2 Long-Term Improvements

Currently, protein folding in Chapter 5 are using conventional mixed strategy. With the knowledge of Chapter 3, the local fitness landscape can be considered introducing into

protein folding. A feature extracted by observing the local fitness, λ, should be defined first. In addition, the experiments in Chapter 5 can be extended to 3-D HP model and functional model proteins.

Furthermore, the differences between mutation operators of different algorithms (Eg. From Immune Algorithm and Differential Evolution) can be compared. Following determination of their own advantages, mutation operators from different algorithms can be potentially combined together with mixed strategy when applying to certain complex local fitness landscapes.

It would be interesting to test the versatility of mixed strategy on other problems and applications, especially other discrete problems. Scheduling could be an potential domain for testing the mixed strategy, including travel salesman problem and flowshop scheduling.

Bibliography

[1] X. Yao, Y. Liu, and G. Lin, "Evolutionary programming made faster," *IEEE Transactions on Evolutionary Computation*, vol. 3, no. 2, pp. 82–102, 1999.

[2] L. J. Fogel, P. J. Angeline, and D. B. Fogel, "An evolutionary programming approach to self-adaptation on finite state machines," in *Proceedings of the Fourth International Conference on Evolutionary Programming*, pp. 355–365, 1995.

[3] C. Y. Lee and X. Yao, "Evolutionary programming using mutations based on the levy probability distribution," *IEEE Transactions on Evolutionary Computation*, vol. 8, no. 1, 2004.

[4] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 67–82, 1997.

[5] K. Chellapilla, "Combining mutation operators in evolutionary programming," *IEEE Transactions on Evolutionary Computation*, vol. 2, no. 3, pp. 91–96, 1998.

[6] J. He and X. Yao, "A game-theoretic approach for designing mixed mutation strategies," in *Proceedings of the First International Conference on Advances in Natural Computation (ICNC 2005)*, Lecture Notes in Computer Science, pp. 279–288, Springer-Verlag GmbH, 2005.

[7] H. Dong, J. He, H. Huang, and W. Hou, "Evolutionary programming using a mixed mutation strategy," *Information Sciences*, vol. 177, no. 1, pp. 312–327, 2007.

[8] H. Zhang and J. Lu, "Adaptive evolutionary programming based on reinforcement learning," *Information Sciences*, vol. 178, no. 4, pp. 971–984, 2008.

[9] Y. Liu, "Operator adaptation in evolutionary programming," in *Advances in Computation and Intelligence*, vol. 4683 of *Lecture Notes in Computer Science*, pp. 90–99, Springer Berlin Heidelberg, 2007.

[10] Z. Michalewicz, *Genetic algorithms + data structures = evolution programs (3rd ed.)*, vol. 1. 1996.

[11] J. H. Holland, *Adaptation in Natural and Artificial Systems: An introductory analysis with applications to biology, control, and artificial intelligence.* 1975.

[12] L. J. Fogel, A. J. Owens, and M. J. Walsh, *Artificial Intelligence Through Simulated Evolution*. John Wiley & Sons, 1966.

[13] T. Bäck, D. B. Fogel, and Z. Michalewicz, *Handbook of Evolutionary Computation*, vol. 2. 1997.

[14] I. Rechenberg, *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Frommann Stuttgart, 1973.

[15] D. B. Fogel, G. B. Fogel, and K. Ohkura, "Multiple-vector self-adaptation in evolutionary algorithms," *BioSystems*, vol. 61, no. 2-3, pp. 155–162, 2001.

[16] M. Dorigo, V. Maniezzo, and A. Colorni, "Positive feedback as a search strategy," tech. rep., 1991.

[17] M. Dorigo, V. Maniezzo, and A. Colorni, "Ant system: Optimization by a colony of cooperating agents," *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. 26, no. 1, pp. 29–41, 1996.

[18] M. Dorigo and C. Blum, "Ant colony optimization theory: A survey," *Theoretical Computer Science*, vol. 344, no. 2-3, pp. 243–278, 2005.

[19] L. N. De Castro and J. Timmis, *Artificial Immune Systems: A New Computational Intelligence Approach*. Springer, 2002.

[20] J. Zheng, Y. Chen, and W. Zhang, "A survey of artificial immune applications," *Artificial Intelligence Review*, vol. 34, no. 1, pp. 19–34, 2010.

[21] J. Timmis, A. Hone, T. Stibor, and E. Clark, "Theoretical advances in artificial immune systems," *Theoretical Computer Science*, vol. 403, no. 1, pp. 11–32, 2008.

[22] E. Hart and J. Timmis, "Application areas of ais: The past, the present and the future," *Applied Soft Computing*, vol. 8, no. 1, pp. 191–201, 2008.

[23] J. Timmis, "Artificial immune systems–today and tomorrow," *Natural Computing*, vol. 6, no. 1, pp. 1–18, 2007.

[24] S. A. Hofmeyr and S. Forrest, "Architecture for an artificial immune system.," *Evolutionary computation*, vol. 8, no. 4, pp. 443–473, 2000.

[25] L. N. de Castro and F. J. von Zuben, "The clonal selection algorithm with engineering applications," in *Proceedings of GECCO*, pp. 36–37, 2000.

[26] L. N. De Castro and F. J. Von Zuben, "Learning and optimization using the clonal selection principle," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 3, pp. 239–251, 2002.

[27] S. F. M. Burnet, *The Clonal Selection Theory of Acquired Immunity.* Vanderbilt University Press, 1959.

[28] V. Cutello, G. Nicosia, and M. Pavone, "Exploring the capability of immune algorithms: A characterization of hypermutation operators," in *Artificial Immune Systems* (G. Nicosia, V. Cutello, P. Bentley, and J. Timmis, eds.), vol. 3239, pp. 263–276, Springer Berlin / Heidelberg, 2004.

[29] A. Eiben, R. Hinterding, and Z. Michalewicz, "Parameter control in evolutionary algorithms," *IEEE Transactions on Evolutionary Computation*, vol. 3, no. 2, 1999.

[30] P. J. Angeline, "Adaptive and self-adaptive evolutionary computations," in *Computational Intelligence: A Dynamic Systems Perspective*, pp. 152–163, IEEE Press, 1995.

[31] K. A. De Jong, *An analysis of the behavior of a class of genetic adaptive systems.* PhD thesis, 1975.

[32] J. D. Schaffer, R. A. Caruana, L. J. Eshelman, and R. Das, "A study of control parameters affecting online performance of genetic algorithms for function optimization," in *Proceedings of the Third International Conference on Genetic Algorithms*, pp. 51–60, Morgan Kaufmann Publishers Inc., 12 1989.

[33] J. Grefenstette, "Optimization of control parameters for genetic algorithms," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 16, no. 1, pp. 122–128, 1986.

[34] H. Mühlenbein, "How genetic algorithms really work: Mutation and hillclimbing," in *Proceedings of the Parallel Problem Solving from Nature 2, (PPSN-II)*, pp. 15–26, Elsevier, 1992.

[35] T. Bartz-Beielstein, *Experimental Research in Evolutionary Computation: The New Experimentalism*. Natural Computing, Springer, 2006.

[36] T. Bartz-Beielstein, C. Lasarczyk, and M. Preuss, "Sequential parameter optimization," in *2005 IEEE Congress on Evolutionary Computation*, vol. 1, 2005.

[37] M. Preuss, "Considerations of budget allocation for sequential parameter optimization (spo)," in *Workshop on Empirical Methods for the Analysis of Algorithms, Proceedings*, pp. 35–40, 2006.

[38] M. Preuss and T. Bartz-Beielstein, "Sequential parameter optimization applied to self-adaptation for binary-coded evolutionary algorithms," *Studies in Computational Intelligence*, vol. 54, pp. 91–119, 2007.

[39] I. Rechenberg, *Evolutionsstrategie '94*. Frommann-Holzboog Verlag, 1994.

[40] C. A. Coello Coello, "Use of a self-adaptive penalty approach for engineering optimization problems," *Computers in Industry*, vol. 41, no. 2, pp. 113–127, 2000.

[41] W. A. De Landgraaf, A. E. Eiben, and V. Nannen, "Parameter calibration using meta-algorithms," in *2007 IEEE Congress on Evolutionary Computation, CEC 2007*, pp. 71–78, 2007.

[42] V. Nannen and A. Eiben, "A method for parameter calibration and relevance estimation in evolutionary algorithms," in *Proceedings of the 8th annual conference on Genetic and evolutionary computation - GECCO '06*, p. 183, ACM Press, 7 2006.

[43] V. Nannen and A. E. Eiben, "Relevance estimation and value calibration of evolutionary algorithm parameters," in *IJCAI International Joint Conference on Artificial Intelligence*, pp. 975–980, 2007.

[44] T. C. Fogarty, "Varying the probability of mutation in the genetic algorithm," in *Proceedings of the Third International Conference on Genetic Algorithms*, pp. 104–109, Morgan Kaufmann Publishers Inc., 12 1989.

[45] J. Hesser and R. Männer, "Towards an optimal mutation probability for genetic algorithms," in *Proceedings of the 1st Workshop on Parallel Problem Solving from Nature*, vol. 496 of *Lecture Notes In Computer Science*, pp. 23–32, Springer-Verlag GmbH, 1991.

[46] T. Bäck and M. Schütz, "Intelligent mutation rate control in canonical genetic algorithms," *Foundations of intelligent systems*, vol. 1079, pp. 158–167, 1996.

[47] J. Joines and C. Houck, "On the use of non-stationary penalty functions to solve nonlinearconstrained optimization problems with ga's," in *Proceedings of the First IEEE Conference on Evolutionary Computation. IEEE World Congress on Computational Intelligence*, pp. 579–584, IEEE, 1994.

[48] I. Rechenberg, *Adaptive Mechanismen in der Biologischen Evolution und ihr Einfluss auf die Evolutionsgeschwindigkeit: Arbeitsbericht*. 1974.

[49] D. Fogel, L. Fogel, and J. Atmar, "Meta-evolutionary programming," in *1991 Conference Record of the Twenty-Fifth Asilomar Conference on Signals, Systems & Computers*, IEEE, 1991.

[50] T. Bäck, "Self-adaptation in genetic algorithms," in *Proceedings of the First European Conference on Artificial Life*, pp. 263–271, MIT Press, 1992.

[51] T. Bäck, "The interaction of mutation rate, selection, and self-adaptation within a genetic algorithm.," in *Parallel Problem Solving from Nature 2, PPSN-II, Brussels, Belgium, September 28-30, 1992*, pp. 87–96, 1 1992.

[52] J. Smith and T. Fogarty, "Self adaptation of mutation rates in a steady state genetic algorithm," in *Proceedings of IEEE International Conference on Evolutionary Computation*, pp. 318–323, IEEE, 1996.

[53] A. Ostermeier, A. Gawelczyk, and N. Hansen, "A derandomized approach to self-adaptation of evolution strategies," *Evolutionary Computation*, vol. 2, no. 4, pp. 369–380, 1994.

[54] H. G. Beyer and D. V. Arnold, "Qualms regarding the optimality of cumulative path length control in csa/cma-evolution strategies.," *Evolutionary computation*, vol. 11, pp. 19–28, 1 2003.

[55] N. Hansen and A. Ostermeier, "Completely derandomized self-adaptation in evolution strategies.," *Evolutionary computation*, vol. 9, no. 2, pp. 159–195, 2001.

[56] P. Koumoutsakos and S. D. Muller, "Flow optimization using stochastic algorithms," *Control of Fluid Flow*, vol. 330, pp. 213–229, 2006.

[57] B. Mersch, T. Glasmachers, P. Meinicke, and C. Igel, "Evolutionary optimization of sequence kernels for detection of bacterial gene starts.," *International journal of neural systems*, vol. 17, no. 5, pp. 369–381, 2007.

[58] H.-g. Beyer and B. Sendhoff, "Covariance matrix adaptation revisited–the cmsa evolution strategy–," in *Proceedings of the 10th international conference on Parallel Problem Solving from Nature: PPSN X*, pp. 123–132, Springer-Verlag, 2008.

[59] O. Kramer, "Evolutionary self-adaptation: A survey of operators and strategy parameters," *Evolutionary Intelligence*, vol. 3, no. 2, pp. 51–65, 2010.

[60] H.-G. Beyer and H.-P. Schwefel, "Evolution strategies – a comprehensive introduction," *Natural Computing*, vol. 1, no. 1, pp. 3–52, 2002.

[61] S. Berlik, *Genetic and Evolutionary Computation – GECCO 2004*, vol. 3102 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 1 2004.

[62] L. Hildebrand, *Asymmetrische Evolutionsstrategien*. PhD thesis, 2003.

[63] O. Kramer, C.-K. T. C.-K. Ting, and H. K. B. H. K. Buning, "A new mutation operator for evolution strategies for constrained problems," *2005 IEEE Congress on Evolutionary Computation*, vol. 3, 2005.

[64] H.-P. P. Schwefel, *Evolution and Optimum Seeking: The Sixth Generation*. John Wiley & Sons, Inc., 8 1993.

[65] J. D. Schaffer and A. Morishima, "An adaptive crossover distribution mechanism for genetic algorithms," in *Proceedings of the Second International Conference on Genetic Algorithms* (J. J. Grefenstette, ed.), pp. 36–40, Lawrence Erlbaum Associates, 1987.

[66] W. M. Spears, "Adapting crossover in evolutionary algorithms," in *Proceedings of the 4th Conference on Evolutionary Programming* (J. R. McDonnell, R. G. Reynolds, and D. B. Fogel, eds.), pp. 367–384, MIT Press, 1995.

[67] O. Kramer and P. Koch, "Self-adaptive partially mapped crossover," in *Proceedings of the 9th annual conference on Genetic and evolutionary computation - GECCO '07*, p. 1523, ACM Press, 7 2007.

[68] G. Reinelt, "Tsplib–a traveling salesman problem library," *INFORMS Journal on Computing*, vol. 3, no. 4, pp. 376–384, 1991.

[69] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, vol. 412 of *Artificial Intelligence*. Addison-Wesley, 1989.

[70] J. H. Holland, *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. MIT Press, 5 1992.

[71] H.-G. Beyer, *The theory of evolution strategies*. Springer Science & Business Media, 3 2013.

[72] L. Davis, "Adapting operator probabilities in genetic algorithms," *Master Thesis, , Department of Artificial Intelligence, Univeristy of Edinburgh*, pp. 61–69, 12 1989.

[73] M. H. Maruo, H. S. Lopes, and M. R. Delgado, "Self-adapting evolutionary parameters : Encoding aspects for combinatorial optimization problems," in *Proceedings of the 5th European Conference on Evolutionary Computation in Combinatorial Optimization*, vol. 3448 of *Lecture Notes in Computer Science*, pp. 154–165, Springer-Verlag GmbH, 2005.

[74] G. R. Harik and D. E. Goldberg, "Learning linkage," in *Foundations of Genetic Algorithms 4* (R. K. Belew and M. D. Vose, eds.), pp. 247–262, Morgan Kaufmann, 1997.

[75] G. R. Harik, F. G. Lobo, and K. Sastry, "Linkage learning via probabilistic modeling in the extended compact genetic algorithm (ecga)," *Studies in Computational Intelligence*, vol. 33, pp. 39–61, 2007.

[76] A. E. Eiben, M. C. Schut, and A. R. de Wilde, "Is self-adaptation of selection pressure and population size possible? - a case study," in *Parallel Problem Solving from Nature-PPSN IX*, vol. IX, pp. 900–909, Springer, 2006.

[77] P. Cowling, G. Kendall, and E. Soubeiga, "A hyperheuristic approach to scheduling a sales summit," *Practice and Theory of Automated Timetabling III*, pp. 176–190, 2001.

[78] H. Fisher and G. L. Thompson, "Probabilistic learning combinations of local job-shop scheduling rules," *Industrial scheduling*, vol. 3, no. 2, pp. 225–251, 1963.

[79] R. H. Storer, S. D. Wu, and R. Vaccari, "New search spaces for sequencing problems with application to job shop scheduling," *Management Science*, vol. 38, no. 10, pp. 1495–1509, 1992.

[80] R. H. Storer, S. D. Wu, and R. Vaccari, "Problem and heuristic space search strategies for job shop scheduling," *ORSA Journal on Computing*, vol. 7, no. 4, pp. 453–467, 1995.

[81] I. P. Norenkov and E. D. Goodma, "Solving scheduling problems via evolutionary methods for rule sequence optimization," in *Soft Computing in Engineering Design and Manufacturing*, pp. 350–355, Springer, 1998.

[82] H. Terashima-Marín, C. J. F. Zárate, P. Ross, and M. Valenzuela-Rendón, "A ga-based method to produce generalized hyper-heuristics for the 2d-regular cutting stock problem," in *Proceedings of the 8th Genetic and evolutionary computation conference (GECCO'06)* (M. Keijzer, ed.), vol. 1 of *GECCO '06*, pp. 591–598, ACM, 2006.

[83] P. Garrido and M. C. Riff, "Collaboration between hyperheuristics to solve strip-packing problems," in *Foundations of fuzzy logic and soft computing*, vol. 4529 LNAI, pp. 698–707, Sringer, 2007.

[84] P. Garrido and M. M.-C. M.-C. Riff, "An evolutionary hyperheuristic to solve strip-packing problems," in *Proceedings of the 8th international conference on Intelligent data engineering and automated learning*, IDEAL'07, pp. 406–415, Springer-Verlag, 2007.

[85] G. Kendall and M. Mohamad, "Channel assignment in cellular communication using a great deluge hyper-heuristic," in *Proceedings - IEEE International Conference on Networks, ICON*, vol. 2, pp. 769–773, IEEE, 2004.

[86] G. Kendall and M. Mohamad, "Channel assignment optimisation using a hyper-heuristic," in *IEEE Conference on Cybernetics and Intelligent Systems, 2004.*, vol. 2, pp. 791–796, IEEE, 2004.

[87] J. R. Koza, "Genetic programming as a means for programming computers by natural selection," *Statistics and Computing*, vol. 4, no. 2, pp. 87–112, 1994.

[88] E. K. Burke, M. Hyde, G. Kendall, G. Ochoa, E. Ozcan, R. Qu, C. E. K. Burke, and O. R. Qu, "A survey of hyper-heuristics," tech. rep., 2009.

[89] R. Battiti and M. Brunato, "Reactive search: Machine learning for memory-based heuristics," tech. rep., 2005.

[90] P. Hansen and N. MladenoviÄĞ, "Variable neighborhood search for the p-median," *Location Science*, vol. 5, no. 4, pp. 207–226, 1997.

[91] P. Moscato and C. Cotta, "A gentle introduction to memetic algorithms," in *Handbook of Metaheuristics*, pp. 105–144, Kluwer Academic Publishers, 2003.

[92] E. Özcan, B. Bilgin, and E. Korkmaz, "A comprehensive analysis of hyper-heuristics," *Intelligent Data Analysis*, vol. 12, no. 1, pp. 3–23, 2008.

[93] C. Grosan and A. Abraham, "Hybrid evolutionary algorithms: methodologies, architectures, and reviews," in *Studies in Computational Intelligence*, vol. 75, pp. 1–17, 2007.

[94] D. B. Fogel, *Evolutionary Compuation: Toward a New Philosophy of Machine Intelligence*, vol. 2010. IEEE Press, 2010.

[95] H. G. Beyer and K. Deb, "On self-adaptive features in real-parameter evolutionary algorithms," *IEEE Transactions on Evolutionary Computation*, vol. 5, no. 3, pp. 250–270, 2001.

[96] S. Meyer-Nieberg and H. G. Beyer, "Self-adaptation in evolutionary algorithms," *Studies in Computational Intelligence*, vol. 54, pp. 47–75, 2007.

[97] D. K. Gehlhaar and D. B. Fogel, "Tuning evolutionary programming for conformationally flexible molecular docking.," in *Evolutionary Programming V: Proceedings of the Fifth Annual Conference on Evolutionary Programming*, pp. 419–429, MIT Press, 1996.

[98] L. Zhuang, H. Dong, J. Jiang, and C. Song, "A genetic algorithm using a mixed crossover strategy," in *Advances in Neural Networks - ISNN 2008* (F. Sun, J. Zhang, Y. Tan, J. Cao, and W. Yu, eds.), vol. 5263 of *Lecture Notes in Computer Science*, pp. 854–863, Springer Berlin Heidelberg, 2008.

[99] M. Pant, M. Ali, and A. Abraham, "Mixed mutation strategy embedded differential evolution," in *2009 IEEE Congress on Evolutionary Computation, CEC 2009*, pp. 1240–1246, IEEE, 2009.

[100] S. Garrett, "Parameter-free, adaptive clonal selection," in *Proceedings of the 2004 Congress on Evolutionary Computation (IEEE Cat. No.04TH8753)*, vol. 1, pp. 1052–1058, IEEE, 2004.

[101] J. Yang, M. Gong, L. Jiao, and L. Zhang, "Improved clonal selection algorithm based on lamarckian local search technique," in *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*, pp. 535–541, IEEE, 2008.

[102] M. Gong, L. Jiao, J. Yang, and F. Liu, "Lamarckian learning in clonal selection algorithm for numerical optimization," *International Journal on Artificial Intelligence Tools*, vol. 19, no. 1, pp. 19–37, 2010.

[103] M. Gong, L. Jiao, and L. Zhang, "Baldwinian learning in clonal selection algorithm for optimization," *Information Sciences*, vol. 180, no. 8, pp. 1218–1236, 2010.

[104] O. Engin and A. Döyen, "A new approach to solve hybrid flow shop scheduling problems by artificial immune system," in *Future Generation Computer Systems*, vol. 20, pp. 1083–1095, 2004.

[105] F. Campelo, F. Guimaraes, H. Igarashi, and J. Ramirez, "A clonal selection algorithm for optimization in electromagnetics," *IEEE Transactions on Magnetics*, vol. 41, no. 5, 2005.

[106] H. Lu and J. Yang, "An improved clonal selection algorithm for job shop scheduling," in *2009 International Symposium on Intelligent Ubiquitous Computing and Education*, pp. 34–37, IEEE, 5 2009.

[107] N. Khilwani, A. Prakash, R. Shankar, and M. K. Tiwari, "Fast clonal algorithm," *Engineering Applications of Artificial Intelligence*, vol. 21, no. 1, pp. 106–128, 2008.

[108] N. Cruz-Cortes, D. Trejo-Perez, and C. A. C. Coello, "Handling constraints in global optimization using an artificial immune system," in *Artificial Immune Systems* (C. Jacob, M. L. Pilat, P. J. Bentley, and J. I. Timmis, eds.), vol. 3627 of

Lecture Notes in Computer Science, pp. 234–247, Springer Berlin Heidelberg, 2005.

[109] A. Acan, "Clonal selection algorithm with operator multiplicity," in *Proceedings of the 2004 Congress on Evolutionary Computation*, vol. 2, pp. 1909–1915, IEEE, 2004.

[110] B. K. Panigrahi, S. R. Yadav, S. Agrawal, and M. K. Tiwari, "A clonal algorithm to solve economic load dispatch," *Electric Power Systems Research*, vol. 77, no. 10, pp. 1381–1389, 2007.

[111] J. M. J. Ma, L. G. L. Gao, and G. S. G. Shi, "An improved immune clonal selection algorithm and its applications for vrp," in *2009 IEEE International Conference on Automation and Logistics*, pp. 2097–2100, IEEE, 2009.

[112] M. K. Tiwari, A. Kumar, and A. R. Mileham, "Determination of an optimal assembly sequence using the psychoclonal algorithm," in *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture*, vol. 219, pp. 137–149, Sage Publications, 2005.

[113] S. Wright, "The roles of mutations, inbreeding, crossbreeding and selection in evolution," in *Proceedings of the 11th International Congress of Genetics*, vol. 1, pp. 356–366, 1932.

[114] C. Reeves, "Fitness landscape," in *Search Methodologies* (E. K. Burke and G. Kendall, eds.), pp. 587–610, Springer US, 2005.

[115] C. R. Reeves and A. V. Eremeev, "Statistical analysis of local search landscapes," *Journal of the Operational Research Society*, vol. 55, pp. 687–693, 7 2004.

[116] L. Shen and J. He, "Evolutionary programming using a mixed strategy adapting to local fitness landscape," *Proceedings of the 2009 UK Workshop on Computational Intelligence*, pp. 92–97, 2009.

[117] L. Shen and J. He, "Evolutionary programming using a mixed strategy with incomplete information," in *2010 UK Workshop on Computational Intelligence (UKCI)*, pp. 1–6, IEEE, 2010.

[118] T. M. Mitchell, *Machine Learning*, vol. 4 of *McGraw-Hill Series in Computer Science*. McGraw-Hill, 1997.

[119] L. Shen and J. He, "A mixed strategy for evolutionary programming based on local fitness landscape," in *IEEE Congress on Evolutionary Computation*, pp. 1–8, IEEE, 7 2010.

[120] J. A. Vrugt, B. A. Robinson, and J. M. Hyman, "Self-adaptive multimethod search for global optimization in real-parameter spaces," *IEEE Transactions on Evolutionary Computation*, vol. 13, no. 2, pp. 243–259, 2009.

[121] B. Haktanirlar Ulutas and S. Kulturel-Konak, "A review of clonal selection algorithm and its applications," *Artificial Intelligence Review*, vol. 36, no. 2, pp. 117–138, 2011.

[122] A. Shmygelska and H. H. Hoos, "An ant colony optimisation algorithm for the 2d and 3d hydrophobic polar protein folding problem," *BMC bioinformatics*, vol. 6, p. 30, 1 2005.

[123] V. Cutello and G. Nicosia, "An immune algorithm for protein structure prediction on lattice models," *Evolutionary Computation , IEEE Transactions on*, vol. 11, no. 1, pp. 101–117, 2007.

[124] M. S. Cheung, L. L. Chavez, and J. N. Onuchic, "The energy landscape for protein folding and possible connections to function," *Polymer*, vol. 45, no. 2, pp. 547–555, 2004.

[125] M. I. Sadowski and D. T. Jones, "The sequence-structure relationship and protein function prediction," *Current Opinion in Structural Biology*, vol. 19, no. 3, pp. 357–362, 2009.

[126] K. F. Lau and K. A. Dill, "Theory for protein mutability and biogenesis," *Proceedings of the National Academy of Sciences*, vol. 87, pp. 638–642, 1 1990.

[127] B. Berger and T. Leighton, "Protein folding in the hydrophobic-hydrophilic (hp) model is np-complete," *Journal of computational biology : a journal of computational molecular cell biology*, vol. 5, pp. 27–40, 1 1998.

[128] A. Szilágyi, J. Kardos, S. Osváth, L. Barna, and P. Závodszky, "Protein folding," in *Handbook of Neurochemistry and Molecular Neurobiology*, pp. 303–343, Springer US, 2007.

[129] N. Krasnogor, B. Blackburne, E. Burke, and J. Hirst, "Muti-mem algorithms for protein structure prediction," in *Parallel Problem Solving from Nature PPSN VII* (J. J. M. Guervós, P. Adamidis, H.-G. Beyer, H.-P. Schwefel, and J.-L. Fernández-Villacañas, eds.), vol. 2439 of *Lecture Notes in Computer Science*, pp. 769–778, Springer Berlin Heidelberg, 10 2002.

[130] M. Socolich, S. W. Lockless, W. P. Russ, H. Lee, K. H. Gardner, and R. Ranganathan, "Evolutionary information for specifying a protein fold," *Nature*, vol. 437, no. 7058, pp. 512–518, 2005.

[131] J. U. Bowie and D. Eisenberg, "Inverted protein structure prediction," *Current Opinion in Structural Biology*, vol. 3, no. 3, pp. 437–444, 1993.

[132] G. A. Cox and R. L. Johnston, "Analyzing energy landscapes for folding model proteins," *The Journal of chemical physics*, vol. 124, p. 204714, 5 2006.

[133] K. F. Lau and K. A. Dill, "A lattice statistical mechanics model of the conformational and sequence spaces of proteins," *Macromolecules*, vol. 22, no. 10, pp. 3986–3997, 1989.

[134] T. C. Beutler and K. A. Dill, "A fast conformational search strategy for finding low energy structures of model proteins," *Protein science : a publication of the Protein Society*, vol. 5, pp. 2037–43, 10 1996.

[135] L. Toma and S. Toma, "Contact interactions method: a new algorithm for protein folding simulations," *Protein science : a publication of the Protein Society*, vol. 5, pp. 147–53, 1 1996.

[136] H.-P. Hsu, V. Mehra, W. Nadler, and P. Grassberger, "Growth algorithms for lattice heteropolymers at low temperatures," *The Journal of Chemical Physics*, vol. 118, p. 444, 8 2003.

[137] R. Unger and J. Moult, "Genetic algorithms for protein folding simulations," *Journal of molecular biology*, vol. 231, pp. 75–81, 5 1993.

[138] R. König and T. Dandekar, "Improving genetic algorithms for protein folding simulations by systematic crossover," *Bio Systems*, vol. 50, pp. 17–25, 4 1999.

[139] F. Liang and W. H. Wong, "Evolutionary monte carlo for protein folding simulations," *The Journal of Chemical Physics*, vol. 115, p. 3374, 8 2001.

[140] T. Jiang, Q. Cui, G. Shi, and S. Ma, "Protein folding simulations of the hydrophobic-hydrophilic model by combining tabu search with genetic algorithms," *The Journal of Chemical Physics*, vol. 119, p. 4592, 8 2003.

[141] D. H. Davies, M. A. Halablab, T. W. K. Young, F. E. G. Cox, and J. Clarke, *Infection and Immunity*. CRC Press, 2002.

[142] S. Flores and J. Smith, "Study of fitness landscapes for the hp model of protein structure prediction," in *The 2003 Congress on Evolutionary Computation, 2003. CEC '03.*, vol. 4, pp. 2338–2345, IEEE, 2003.

[143] G. A. Cox, T. V. Mortimer-Jones, R. P. Taylor, and R. L. Johnston, "Development and optimisation of a novel genetic algorithm for studying model protein folding," *Theoretical Chemistry Accounts*, vol. 112, pp. 163–178, 6 2004.

[144] T. N. Bui and G. Sundarraj, "An efficient genetic algorithm for predicting protein tertiary structures in the 2d hp model," in *Proceedings of the 2005 conference on Genetic and evolutionary computation - GECCO '05*, p. 385, ACM Press, 6 2005.

[145] C. M. Johnson and A. Katikireddy, "A genetic algorithm with backtracking for protein structure prediction," in *Proceedings of the 8th annual conference on Genetic and evolutionary computation - GECCO '06*, p. 299, ACM Press, 7 2006.

[146] R. Storn and K. Price, "Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces," *Journal of Global Optimization*, vol. 11, no. 4, pp. 341–359, 1997.